Lab Exercises

For

Java

Table of Contents

1 Java Basics

1.1 Language Fundamentals

1) The following program has several errors. Modify it so that it will compile and run without errors.

```
import java.util.*;

package example.counter;

public class Exercise1 {
    int counter;

    void main(String[] args) {
        Exercise1 instance = new Exercise1();
        instance.go();
    }

    public void go() {
        int sum;
        int i = 0;
        while (i<100) {
            if (i == 0) sum = 100;
            sum = sum + i;
            i++;
        }
        System.out.println(sum);
    }
}
```

2) The following program has several errors. Modify it so that it will compile and run without errors.

```
// Filename: Temperature.java
PUBLIC CLASS temperature {
    PUBLIC void main(string args) {
        double fahrenheit = 62.5;
        */ Convert /*
        double celsius = f2c(fahrenheit);
        System.out.println(fahrenheit + 'F = ' + celsius + 'C');
    }

    double f2c(float fahr) {
        RETURN (fahr - 32) * 5 / 9;
    }
}
```

1.2 Operators and Assignments

1) (Converting feet into meters) Write a program that reads a number in feet, converts it to meters, and displays the result. One foot is 0.305 meters.

2) (Converting pounds into kilograms) Write a program that converts pounds into kilograms. The program prompts the user to enter a number in pounds, converts it to kilograms, and displays the result. One pound is 0.454 kilograms.

3) The program below is supposed to calculate and show the time it takes for light to travel from the sun to the earth. It contains some logical errors. Fix the program so that it will compile and show the intended value when run.

```
// Filename: Sunlight.java
public class Sunlight {
    public static void main(String[] args) {
        // Distance from sun (150 million kilometers)
        int kmFromSun = 150000000;

        int lightSpeed = 299792458; // meters per second

        // Convert distance to meters.
        int mFromSun = kmFromSun * 1000;

        int seconds = mFromSun / lightSpeed;

        System.out.print("Light will use ");
        printTime(seconds);
        System.out.println(" to travel from the sun to the earth.");
    }

    public static void printTime(int sec) {
        int min = sec / 60;
        sec = sec - (min*60);
        System.out.print(min + " minute(s) and " + sec + " second(s)");
    }
}
```

4) Create two methods that both take an int as an argument and return a String object representing the binary representation of the integer. Given the argument 42, it should return "101010". The first method should calculate the binary representation manually, and the other should use the functionality available in the Java class libraries.

5) Create a program that will print every other argument given on the command line. If the program was executed with the following on the command line,

```
java ArgumentSkipper one two three a b c d
```

the program would print

```
one
three
b
```

```
d
```

Consider how your program would operate when no arguments are given.

6) (Calculating the future investment value) Write a program that reads in investment amount, annual interest rate, and number of years, and displays the future investment value using the following formula:

```
futureInvestmentValue =
  investmentAmount x (1 + monthlyInterestRate)numberOfYears*12
```

For example, if you entered amount 1000, annual interest rate 3.25%, and number of years 1, the future investment value is 1032.98.

1.3 Control Flow

1) A large company pays its salespeople on a commission basis. The salespeople receive Rs200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells Rs 5,000 worth of merchandise in a week receives Rs200 plus 9% of Rs5,000, or a total of Rs650. You have been supplied with a list of the items sold by each salesperson. The values of these items are as follows:

Item	Value
1	239.99
2	129.75
3	99.95
4	350.89

Develop a Java application that inputs one salesperson's items sold for last week and calculates and displays that salesperson's earnings. There is no limit to the number of items that can be sold by a salesperson.

2) Develop a Java application that will determine the gross pay for each of three employees. The company pays straight time for the first 40 hours worked by each employee and time and a half for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee and should determine and display the employee's gross pay.

3) (Conversion from kilograms to pounds) Write a program that displays the following table (note that 1 kilogram is 2.2 pounds):

```
Kilograms Pounds
1         2.2
3         6.6
```

```
...
197         433.4
199         437.8
```

4) Write an application that calculates the squares and cubes of the numbers from 0 to 10 and prints the resulting values in table format, as shown below. [Note: This program does not require any input from the user.]

```
number   square   cube
0        0        0
1        1        1
2        4        8
3        9        27
4        16       64
5        25       125
6        36       216
7        49       343
8        64       512
9        81       729
10       100      1000
```

5) Write a Java application that uses looping to print the following table of values:

N	10*N	100*N	1000*N
1	10	100	1000
2	20	200	2000
3	30	300	3000
4	40	400	4000
5	50	500	5000

6) (Finding numbers divisible by 5 and 6) Write a program that displays all the numbers from 100 to 1000, ten per line, that are divisible by 5 and 6.

7) (Finding numbers divisible by 5 or 6, but not both) Write a program that displays all the numbers from 100 to 200, ten per line, that are divisible by 5 or 6, but not both.

8) (Finding the smallest n such that n2 > 12000) Use a while loop to find the smallest integer n such that n is greater than 12,000.

9) (Finding the largest n such that n3 < 12000) Use a while loop to find the largest integer n such that n is less than 12,000.

10) (Displaying the ACSII character table) Write a program that prints the characters in the ASCII character table from '!' to '~'. Print ten characters per line.

1.4 Methods

1) (Averaging an array) Write two overloaded methods that return the average of an array with the following headers:

```
public static int average(int[] array);
public static double average(double[] array);
```

Use {1, 2, 3, 4, 5, 6} and {6.0, 4.4, 1.9, 2.9, 3.4, 3.5} to test the methods.

2) (Finding the smallest element) Write a method that finds the smallest element in an array of integers. Use {1, 2, 4, 5, 10, 100, 2, –22} to test the method.

3) (Finding the index of the smallest element) Write a method that returns the index of the smallest element in an array of integers. If there are more than one such elements, return the smallest index. Use {1, 2, 4, 5, 10, 100, 2, –22} to test the method.

4) (Computing commissions) Write a method that computes the commission, using the scheme given below:

Sales Amount	Commission Rate
Rs0.01-Rs5,000	8 percent
Rs5,000.01- Rs10,000	10 percent
Rs10,000.01 and above	12 percent

Finding the Sales Amount. The header of the method is:

```
public static double computeCommission(double salesAmount)
```

Write a test program that displays the following table:

```
SalesAmount    Commission
 10000           900.0
 15000          1500.0
   ...
 95000         11100.0
100000         11700.0
```

1.5 Object Oriented Programming

1) (Savings Account Class) Create class `SavingsAccount`. Use a `static` variable `annualInterestRate` to store the annual interest rate for all account holders. Each object of the class contains a `private` instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by

annualInterestRate divided by 12 this interest should be added to savingsBalance. Provide a static method modifyInterestRate that sets the annualInterestRate to a new value. Write a program to test class SavingsAccount. Instantiate two savingsAccount objects, saver1 and saver2, with balances of Rs2000.00 and Rs3000.00, respectively. Set annualInterestRate to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the annualInterestRate to 5%, calculate the next month's interest and print the new balances for both savers.

2) (Rational Numbers) Create a class called Rational for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the private instance variables of the class the numerator and the denominator. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form. The fraction

```
2/4
```

is equivalent to 1/2 and would be stored in the object as 1 in the numerator and 2 in the denominator. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform each of the following operations:

a. Add two Rational numbers: The result of the addition should be stored in reduced form.
b. Subtract two Rational numbers: The result of the subtraction should be stored in reduced form.
c. Multiply two Rational numbers: The result of the multiplication should be stored in reduced form.
d. Divide two Rational numbers: The result of the division should be stored in reduced form.
e. Print Rational numbers in the form a/b, where a is the numerator and b is the denominator.
f. Print Rational numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

3) (The Rectangle class) Design a class named Rectangle to represent a rectangle. The class contains:
Two double data fields named width and height that specify the width and height of the rectangle. The default values are 1 for both width and height.

A string data field named color that specifies the color of a rectangle. Hypothetically, assume that all rectangles have the same color. The default color is white.

A no-arg constructor that creates a default rectangle.

A constructor that creates a rectangle with the specified width and height.

The accessor and mutator methods for all three data fields.

A method named `getArea()` that returns the area of this rectangle.

A method named `getPerimeter()` that returns the perimeter.

Write a test program that creates two `Rectangle` objects. Assign width 4 and height 40 to the first object and width 3.5 and height 35.9 to the second object. Assign color red to all `Rectangle` objects. Display the properties of both objects and find their areas and perimeters.

4) (The `Fan` class) Design a class named `Fan` to represent a fan. The class contains:

- Three constants named `SLOW`, `MEDIUM`, and `FAST` with values `1`, `2`, and `3` to denote the fan speed.
- An `int` data field named `speed` that specifies the speed of the fan (default `SLOW`).
- A `boolean` data field named `on` that specifies whether the fan is on (default `false`).
- A `double` data field named `radius` that specifies the radius of the fan (default `5`).
- A string data field named `color` that specifies the color of the fan (default `blue`).
- A no-arg constructor that creates a default fan.
- The accessor and mutator methods for all four data fields.
- A method named `toString()` that returns a string description for the fan. If the fan is on, the method returns the fan speed, color, and radius in one combined string. If the fan is not on, the method returns fan color and radius along with the string `"fan is off"` in one combined string.

Implement the class. Write a test program that creates two `Fan` objects. Assign maximum speed, radius `10`, color `yellow`, and turn it on to the first object. Assign medium speed, radius `5`, color `blue`, and turn it off to the second object. Display the objects by invoking their `toString` method.

5) (The `Stock` class) Design a class named `Stock` that contains:

- A string data field named `symbol` for the stock's symbol.
- A string data field named `name` for the stock's name.
- A `double` data field named `previousClosingPrice` that stores the stock price for the previous day.
- A `double` data field named `currentPrice` that stores the stock price for the current time.
- A constructor that creates a stock with specified symbol and name.
- The accessor methods for all data fields.
- The mutator methods for `previousClosingPrice` and `currentPrice`.

- A method named `changePercent()` that returns the percentage changed from `previousClosingPrice` to `currentPrice`.

Implement the class. Write a test program that creates a `Stock` object with the stock symbol SBIN, the name State Bank of India, and the previous closing price of `1840`. Set a new current price to 1810 and display the price-change percentage.

6) (The `Time` class) Design a class named `Time`. The class contains:

- Data fields `hour`, `minute`, and `second` that represents a time.
- A no-arg constructor that creates a `Time` object for the current time. (The data fields value will represent the current time)
- A constructor that constructs a `Time` object with a specified elapse time since the middle night, January 1, 1970 in milliseconds. (The data fields value will represent this time.)
- Three get methods for the data fields `hour`, `minute`, and `second`, respectively.

Implement the class. Write a test program that creates two `Time` objects (using `new Time()` and `new Time(555550000)`) and display their hour, minute, and second.

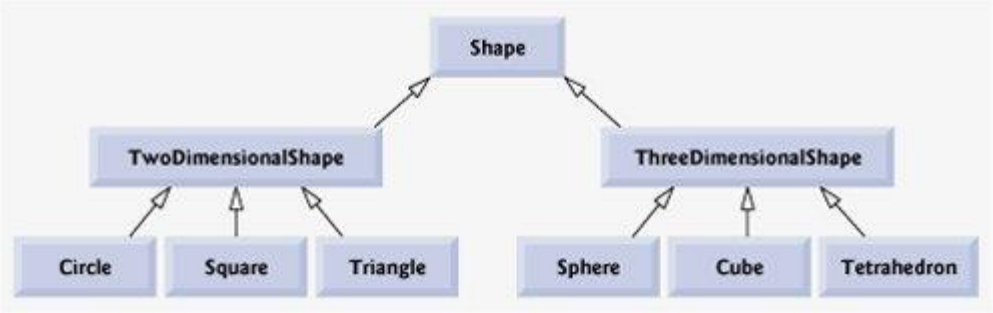
7) (The `MyInteger` class) Design a class named `MyInteger`. The class contains:

- An `int` data field named `value` that stores the `int` value represented by this object.
- A constructor that creates a `MyInteger` object for the specified `int` value.
- A get method that returns the `int` value.
- Methods `isEven()`, `isOdd()`, and `isPrime()` that return `true` if the value is even, odd, or prime, respectively.
- Static methods `isEven(int)`, `isOdd(int)`, and `isPrime(int)` that return `true` if the specified value is even, odd, or prime, respectively.
- Static methods `isEven(MyInteger)`, `isOdd(MyInteger)`, and `isPrime(MyInteger)` that return `true` if the specified value is even, odd, or prime, respectively.
- Methods `equals(int)` and `equals(MyInteger)` that return `true` if the value in the object is equal to the specified value.
- A static method `parseInt(int)` that converts a string to an `int` value.

Implement the class. Write a client program that tests all methods in the class.

8) (Shape Hierarchy) Implement the `Shape` hierarchy shown in the figure given below. Each `TwoDimensionalShape` should contain method `getArea` to calculate the area of the two-dimensional shape. Each `ThreeDimensionalShape` should have methods `getArea` and `getVolume` to calculate the surface area and volume, respectively, of the three-dimensional shape. Create a program that uses an array of `Shape` references to objects of each concrete class in the hierarchy. The program should print a text description of the

object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a `TwoDimensionalShape` or a `ThreeDimensionalShape`. If a shape is a `TwoDimensionalShape`, display its area. If a shape is a `ThreeDimensionalShape`, display its area and volume.



9) (The `Person`, `Student`, `Employee`, `Faculty`, and `Staff` classes) Design a class named `Person` and its two subclasses named `Student` and `Employee`. Make `Faculty` and `Staff` subclasses of `Employee`. A person has a name, address, phone number, and email address. A student has a class status (freshman, sophomore, junior, or senior). Define the status as a constant. An employee has an office, salary, and date-hired. Define a class named `MyDate` that contains the fields `year`, `month`, and `day`. A faculty member has office hours and a rank. A staff member has a title. Override the `toString` method in each class to display the class name and the person's name.

Implement the classes. Write a test program that creates a `Person`, `Student`, `Employee`, `Faculty`, and `Staff`, and invokes their `toString()` methods.

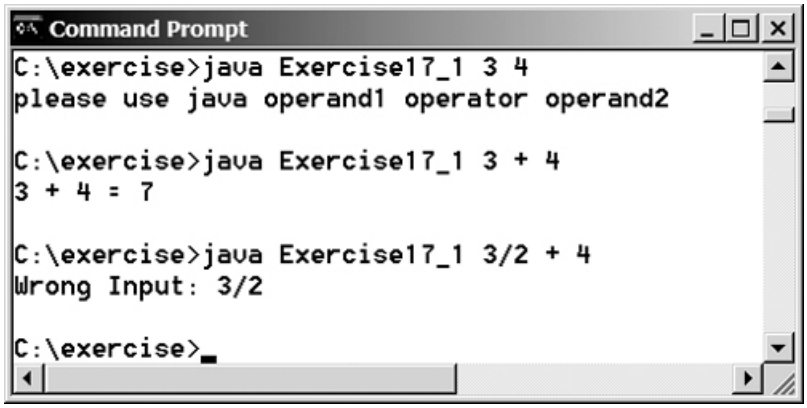
1.6 Exception Handling

1) (`NumberFormatException`) The following program, Calculator.java, is a simple command-line calculator.

```
1 public class Calculator {
2   /** Main method */
3   public static void main(String[] args) {
4     // Check number of strings passed
5     if (args.length != 3) {
6       System.out.println(
7         "Usage: java Calculator operand1 operator operand2");
8       System.exit(0);
9     }
10
11    // The result of the operation
12    int result = 0;
13
```

```
14      // Determine the operator
15      switch (args[1].charAt(0)) {
16      case '+': result = Integer.parseInt(args[0]) +
17                         Integer.parseInt(args[2]);
18                break;
19      case '-': result = Integer.parseInt(args[0]) -
20                         Integer.parseInt(args[2]);
21                break;
22      case '*': result = Integer.parseInt(args[0]) *
23                         Integer.parseInt(args[2]);
24                break;
25      case '/': result = Integer.parseInt(args[0]) /
26                         Integer.parseInt(args[2]);
27    }
28
29    // Display result
30    System.out.println( args[0] + ' ' + args[1] + ' ' args[2]
31      + " = " + result);
32  }
33 }
```

Note that the program terminates if any operand is non-numeric. Write a program with an exception handler that deals with non-numeric operands; then write another program without using an exception handler to achieve the same objective. Your program should display a message that informs the user of the wrong operand type before exiting (see figure given below).



2) (Catching Exceptions Using Class Exception) Write a program that demonstrates how various exceptions are caught with

```
catch ( Exception exception )
```

This time, define classes `ExceptionA` (which inherits from class Exception) and `ExceptionB` (which inherits from class `ExceptionA`). In your program, create try blocks

that throw exceptions of types `ExceptionA`, `ExceptionB`, `NullPointerException` and `IOException`. All exceptions should be caught with catch blocks specifying type Exception.

3) (Order of catch Blocks) Write a program that shows that the order of catch blocks is important. If you try to catch a superclass exception type before a subclass type, the compiler should generate errors.

4) (Constructor Failure) Write a program that shows a constructor passing information about constructor failure to an exception handler. Define class `SomeException`, which throws an Exception in the constructor. You program should try to create an object of type `SomeException`, and catch the exception that is thrown from the constructor.

5) (Rethrowing Exceptions) Write a program that illustrates rethrowing an exception. Define methods `someMethod` and `someMethod2`. Method `someMethod2` should initially throw an exception. Method `someMethod` should call `someMethod2`, catch the exception and rethrow it. Call `someMethod` from method main, and catch the rethrown exception. Print the stack trace of this exception.

6) (Catching Exceptions Using Outer Scopes) Write a program showing that a method with its own Try block does not have to catch every possible error generated within the try. Some exceptions can slip through to, and be handled in, other scopes.

2 Core Java

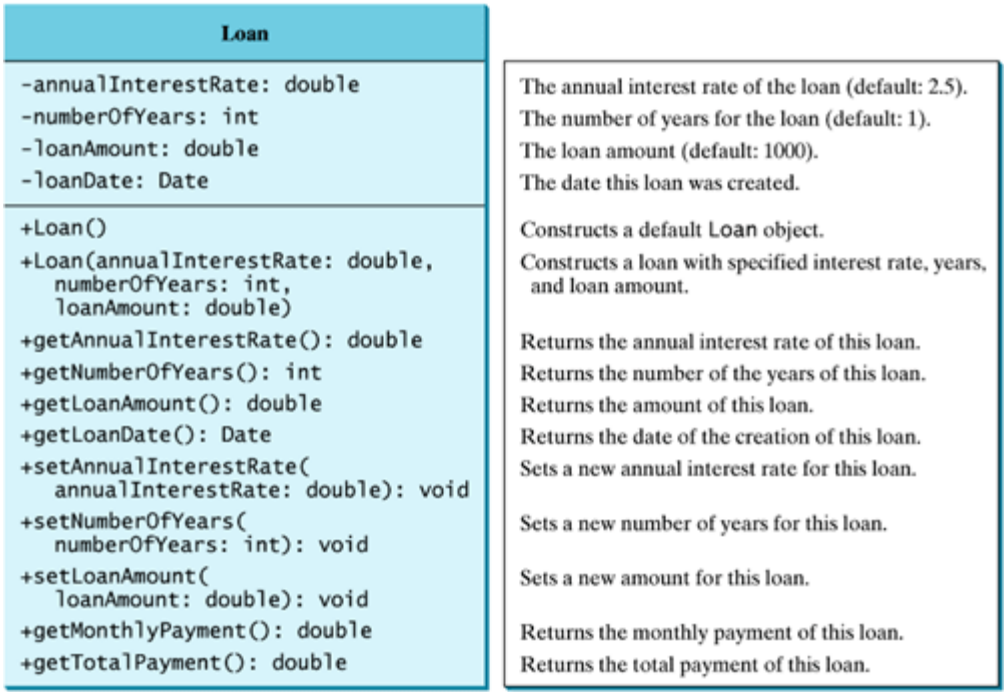
IO

1) (Creating a text file) Write a program to create a file named `Numbers.txt` if it does not exist. Append new data to it if the file already exists. Write one hundred integers created randomly into the file using text I/O. Integers are separated by a space.

2) (Summing all the integers in a binary data file) Suppose a binary data file named `BinaryNumbers.dat` has been created using `writeInt(int)` in `DataOutputStream`. The file contains an unspecified number of integers. Write a program to find the total of integers.

3) (Storing objects and arrays into a file) Write a program that stores an array of five `int` values `1`, `2`, `3`, `4` and `5`, a `Date` object for current time, and a double value `5.5` into the file named `Objects.dat`.

4) (Storing Loan objects) The `Loan` class, introduced below.



Write the `Loan` class to implement `Serializable`. Write a program that creates five `Loan` objects and stores them in a file named `Loan.dat`.

2.1 Collections

1) Write a program that reads in a series of first names and stores them in a `LinkedList`. Do not store duplicate names. Allow the user to search for a first name.

2) Write a program to count the number of occurrences of each letter in a given sentence. For example, the string "HELLO THERE" contains two Hs, three Es, two Ls, one O, one T and one R. Display the results.

3) (Performing set operations on array lists) Create two array lists {"George", "Jim", "John", "Blake", "Kevin", "Michael"} and {"George", "Katie", "Kevin", "Michelle", "Ryan"}, and find their union, difference, and intersection. (You may clone the lists to preserve the original lists from being changed by these methods.)

4) (Performing set operations on hash sets) Create two hash sets {"George", "Jim", "John", "Blake", "Kevin", "Michael"} and {"George", "Katie", "Kevin", "Michelle", "Ryan"}, and find their union, difference, and intersection. (You may clone the sets to preserve the original sets from being changed by these set methods.)

2.2 JDBC

1) Define a complete query application for the books database. Provide the following predefined queries:

   a. Select all authors from the `authors` table.
   b. Select all publishers from the `publishers` table.
   c. Select a specific author and list all books for that author. Include the title, year and ISBN. Order the information alphabetically by the author's last name and first name.
   d. Select a specific publisher and list all books published by that publisher. Include the title, year and ISBN. Order the information alphabetically by title.
   e. Provide any other queries you feel are appropriate.

Note: Define the required tables with necessary fields. Do the same for Java classes.