6.7: (Financial application: compute the future investment value) Write a method that computes future investment value at a given interest rate for a specified number of years. The future investment is determined using the formula in Programming Exercise 2.21.

Use the following method header:

public static double futureInvestmentValue(double investmentAmount, double monthlyInterestRate, int years)

For example, futureInvestmentValue(10000, 0.05/12, 5) returns

12833.59.

Write a test program that prompts the user to enter the investment amount (e.g.,

1000) and the interest rate (e.g., 9%) and prints a table that displays future value

for the years from 1 to 30.

Code:

import java.util.Scanner;

public class Ex7 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in); // Creating a java scanner

final int NUMBER\_OF\_YEARS = 30; // Number of years

// Prompt the user to enter the investment amount and interest rate

System.out.print("\nThe amount invested: ");

double amount = input.nextDouble();

System.out.print("Annual interest rate: ");

double annualInterestRate = input.nextDouble();

// Get monthly interest rate

double monthlyInterestRate = annualInterestRate / 1200;

// Print a table that displays future value for the years from 1 to 30

System.out.println("Years Future Value");

for (int years = 1; years <= NUMBER\_OF\_YEARS; years++) {

System.out.printf("%-10d", years);

System.out.printf("%11.2f\n",

futureInvestmentValue(amount, monthlyInterestRate, years));

}

}

public static double futureInvestmentValue(

double investmentAmount, double monthlyInterestRate, int years) {

return investmentAmount \* Math.pow(1 + monthlyInterestRate, years \* 12);

}

}

Output: **Oh to live in a world where an annual interest rate is only 2.5 :)**

The amount invested: 1000

Annual interest rate: 2.5

Years Future Value

1 1025.29

2 1051.22

3 1077.80

4 1105.06

5 1133.00

6 1161.65

7 1191.03

8 1221.15

9 1252.03

10 1283.69

11 1316.15

12 1349.44

13 1383.56

14 1418.55

15 1454.42

16 1491.20

17 1528.91

18 1567.58

19 1607.22

20 1647.86

21 1689.54

22 1732.26

23 1776.07

24 1820.98

25 1867.03

26 1914.25

27 1962.65

28 2012.29

29 2063.17

30 2115.35

6.10 (Use the isPrime Method) Listing 6.7, PrimeNumberMethod.java, provides the isPrime(int number) method for testing whether a number is prime. Use this method to find the number of prime numbers less than 10000.

Method class code:

public class PrimeNumberMethod {

public static void main(String[] args) {

System.out.println("The first 50 prime numbers are \n");

printPrimeNumbers(50);

}

public static void printPrimeNumbers(int numberOfPrimes) {

final int NUMBER\_OF\_PRIMES\_PER\_LINE = 10; // Display 10 per line

int count = 0; // Count the number of prime numbers

int number = 2; // A number to be tested for primeness

while (count < numberOfPrimes) {

// Print the prime number and increase the count

if (isPrime(number)) {

count++; // Increase the count

if (count % NUMBER\_OF\_PRIMES\_PER\_LINE == 0) {

// Print the number and advance to the new line

System.out.printf("%-5s\n", number);

}

else

System.out.printf("%-5s", number);

}

// Check whether the next number is prime

number++;

}

}

public static boolean isPrime(int number) {

for (int divisor = 2; divisor <= number/2; divisor++) {

if (number % divisor == 0) { // If true, number is not prime

return false; // Number is not a prime

}

}

return true; // Number is prime

}

}

Ex 10 Code with implemented Prime Number Method:

public class Ex10 {

public static void main(String[] args) {

int numberOfPrimes = 0; // Counts the number of prime numbers

// Find the number of prime numbers less than 10000

for (int i = 1; i < 1000; i++) {

if (PrimeNumberMethod.isPrime(i))

numberOfPrimes++; // Increament

}

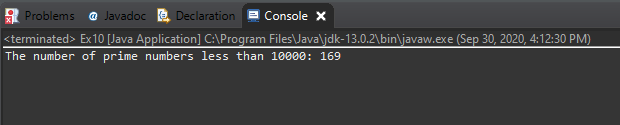
System.out.println(

"The number of prime numbers less than 10000: " + numberOfPrimes);

}

}

Output:



6.22 (Math: approximate the square root) There are several techniques for implementing the sqrt method in the Math class. One such technique is known as the Babylonian method. It approximates the square root of a number, n, by repeatedly performing a calculation using the following formula:

nextGuess = (lastGuess + n / lastGuess) / 2

When nextGuess and lastGuess are almost identical, nextGuess is the approximated square root. The initial guess can be any positive value (e.g., 1). This value will be the starting value for lastGuess. If the difference between nextGuess and lastGuess is less than a very small number, such as 0.0001,you can claim that nextGuess is the approximated square root of n. If not, next-Guess becomes lastGuess and the approximation process continues. Implement the following method that returns the square root of n.

public static double sqrt(long n)

Code:

import java.util.Scanner;

public class Ex22 {

public static void main(String[] args) {

Scanner input = new Scanner(System.in); // Create a Scanner

// Prompt the user to enter a number

System.out.print("Enter a number: ");

long number = input.nextLong();

// Display the square root

System.out.println(

"The approximated square root of " + number + " is: " + sqrt(number));

}

/\*\* Method squrt approximates the square root of n \*/

public static double sqrt(long n) {

long lastGuess = 1;

long nextGuess = (lastGuess + n / lastGuess) / 2;

while (nextGuess - lastGuess > 0.0001) {

lastGuess = nextGuess;

nextGuess = (lastGuess + n / lastGuess) / 2;

}

lastGuess = nextGuess;

return nextGuess = (lastGuess + n / lastGuess) / 2;

}

}

Output:

