1. (5 Points) What are syntax errors, run time errors and logic errors? Explain and demo

|  |
| --- |
| **Syntax Errors** - The error in the source code of a program is called syntax error. The syntax in any program should be correct so that we won't have any compilation errors. Syntax errors are nothing but that grammatical mistakes, may be limited to a single character such as semicolon, bracket etc.  **Logic Error** – The bug or error in the source code is called logic error. This may result in the incorrect or unexpected behavior. It is a type of runtime error which may give a wrong output. For example, assigning a value to the wrong variable may cause a series of unexpected program errors. Multiplying two numbers instead of adding them together may also produce unwanted results.  **Runtime Error** – The error that occurs while the program is running. Program crash is one of the most noticeable type in runtime error. Examples include dividing by zero, referencing missing files, calling invalid functions, or not handling certain [input](https://techterms.com/definition/input) correctly. |

1. (5 Points) Design a class named Fan to represent a fan. The class contains:
2. Three constants named **SLOW, MEDIUM, a**nd **FAST** with the values **1, 2,** and **3** to denote the fan speed.
3. A private **int** data field named **speed** that specifies the speed of the fan
4. A private **boolean** data field named **on** that specifies whether the fan is on
5. A private **double** data field named **radius** that specifies the radius of the fan
6. A **string** data field named **color** that specifies the color of the fan
7. The accessor and mutator methods for all four data fields.
8. A no-arg constructor that creates a default fan.
9. 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 the fan color and radius along with the string “fan is off” in one combined string.
10. Write a Driver class to test all the methods of Fan class and provide sample input and output

|  |
| --- |
| **Fan Class**  package fan;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class Fan {  private static final int slow = 1;  private static final int medium = 2;  private static final int fast = 3;  private int speed;  private boolean on;  private double radius;  private String color;  /\*\*  \*  \* @return speed  \*/  public int getSpeed() {  return speed;  }  /\*\*  \*  \* @param speed  \*/  public void setSpeed(int speed) {  this.speed = speed;  }  /\*\*  \*  \* @returns a boolean value  \*/  public boolean isOn() {  return on;  }  /\*\*  \*  \* @param on  \*/  public void setOn(boolean on) {  this.on = on;  }  /\*\*  \*  \* @return radius  \*/  public double getRadius() {  return radius;  }  /\*\*  \*  \* @param radius  \*/  public void setRadius(double radius) {  this.radius = radius;  }  /\*\*  \*  \* @return color  \*/  public String getColor() {  return color;  }  /\*\*  \*  \* @param color  \*/  public void setColor(String color) {  this.color = color;  }  /\*\*  \* A constructor with no parameters  \*/  public Fan() {  }  @Override  public String toString() {  if (on) {  return "Fan{" + "speed=" + speed + ", radius=" + radius + ", color=" + color + '}';  }  return "Fan is Off{" + "radius=" + radius + ", color=" + color + '}';  }  }  **Fan Driver Class**  package fan;  import java.util.\*;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class FanDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  /\*\*  \* Created an object  \*/  Fan fan = new Fan();  /\*\*  \* set values for Speed Radius Color On  \*/  fan.setSpeed(3);  fan.setRadius(10);  fan.setColor("peach");  fan.setOn(true);  /\*\*  \* Here we refer to the toString() from Fan Class  \*/  System.out.println(fan.toString());  /\*\*  \* set values for Speed Radius Color  \*/  fan.setSpeed(2);  fan.setRadius(10);  fan.setColor("lavender");  fan.setOn(false);  /\*\*  \* Here we refer to the toString() from Fan Class On  \*/  System.out.println(fan.toString());  }  }  **Output**  Fan{speed=3, radius=10.0, color=peach}  Fan is Off{radius=10.0, color=lavender} |

1. (5 Points) Write the output for below code and explain the concept of equality comparison? Use figures to explain the references.

|  |
| --- |
| Rectangle box1 = new Rectangle(10, 20, 10, 10);  Rectangle box2 = new Rectangle(10, 20, 10, 10);  Rectangle box3 = box1;  Rectangle box4;  System.out.println(box1 == box2);  System.out.println(box1.equals(box2));  System.out.println(box2 == box3);  System.out.println(box2.equals(box3));  System.out.println(box1 == box3);  box3 = new Rectangle(10, 20, 10, 10);  System.out.println(box1 == box3);  System.out.println(box1.equals(box3));  String s1 = new String("Java");  String s2 = s1;  String s3 = new String(s2);  String s4 = "Java";  System.out.println(s1 == s2);  System.out.println(s1.equals(s2));  System.out.println(s1 == s3);  System.out.println(s1.equals(s3));  System.out.println(s1 == s4);  System.out.println(s1.equals(s4));  System.out.println(s2 == s3);  System.out.println(s2.equals(s3));  System.out.println(s2 == s4);  System.out.println(s2.equals(s4));  System.out.println(s3 == s4);  System.out.println(s3.equals(s4));  Answers :  false  true  false  true  true  false  true  true  true  false  true  false  true  false  true  false  true  false  true |

1. (5 Points) Write a program that prompts the user to enter a Social Security number in the format DDD-DD-DDDD, where D is a digit. Your program should check whether the input is valid. Provide sample input and output.

|  |
| --- |
| **Ssn Driver**  package ssn;  import java.util.Scanner;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class SsnDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  /\*\*  \* User Input is taken  \*/  System.out.println("Enter a SSN:");  Scanner scan = new Scanner(System.in);  String ssn = scan.nextLine();  /\*\*  \* trim the extra spaces  \*/  String a = ssn.trim();  /\*\*  \* Enters the loop only if it satisfies the conditions such as length of  \* string and splitting the characters with '-'  \*/  /\*\*  \* Divides the given string after third digits and also after fifth digit  \*/  if (a.length() == 11 && a.charAt(3) == '-' && ssn.charAt(6) == '-') {  System.out.println(a + " is an valid social security number");  } else {  System.out.println(a + "is an invalid social security number");  }  }  }  **Output :**  Enter a SSN:  235-45-6789   * + 1. an valid social security number |

1. (5 Points) Write a program that displays all possible combinations for picking two numbers from integers 1 to 7. Display total number of combinations.

|  |
| --- |
| Combinations Class  package combinations;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class Combinations {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  /\*\*  \* Checks for values and increments and as long as it satisfies the  \* condition prints the values  \*/  int count = 0;  for (int i = 1; i < 8; i++) {  for (int j = i + 1; j < 8; j++) {  System.out.println(i + " , " + j);  count++;  }  }  System.out.println("The total number of all combinations is " + count);  }  }  **Output :**  1 , 2  1 , 3  1 , 4  1 , 5  1 , 6  1 , 7  2 , 3  2 , 4  2 , 5  2 , 6  2 , 7  3 , 4  3 , 5  3 , 6  3 , 7  4 , 5  4 , 6  4 , 7  5 , 6  5 , 7  6 , 7  The total number of all combinations is 21 |

1. (5 Points) Write a program that prompts the user to enter the length from the center of a pentagon to a vertex and computes the area of the pentagon, as shown in the following figure.

r

The formula for computing the area of a pentagon is where s is the length of a side. The side can be computed using the formula **,** where r is the length from the center of a pentagon to a vertex. Round up two digits after the decimal point.

|  |
| --- |
| Pentagon Class  package pentagon;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class Pentagon {  private double side;  private double area;  private double radius;  /\*\*  \*  \* @param radius  \*/  public Pentagon(double radius) {  this.radius = radius;  }  /\*\*  \*  \* @return  \*/  public double getSide() {  return side;  }  /\*\*  \*  \* @param side  \*/  public void setSide(double side) {  this.side = side;  }  /\*\*  \*  \* @return  \*/  public double getArea() {  return area;  }  /\*\*  \*  \* @param area  \*/  public void setArea(double area) {  this.area = area;  }  /\*\*  \*  \* @return radius  \*/  public double getRadius() {  return radius;  }  /\*\*  \*  \* @param radius  \*/  public void setRadius(double radius) {  this.radius = radius;  }  /\*\*  \*  \* @param side  \* @return side  \*/  public double getSideOf(double side) {  side = (2 \* getRadius()) \* Math.sin(Math.PI / 5);  return side;  }  /\*\*  \*  \* @param area  \* @return side  \*/  public double getAreaOf(double area) {  area = (5 \* Math.pow(getSideOf(side), 2)) / (4 \* Math.tan(Math.PI / 5));  return area;  }  @Override  public String toString() {  double d = Math.round(getAreaOf(area) \* 100.00) / 100.00;  String s1 = Double.toString(d);  return s1;  }  }  Pentagon Driver Class  package pentagon;  import java.util.Scanner;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class PentagonDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  double radius;  /\*\*  \* Reads input values from user  \*/  Scanner scan = new Scanner(System.in);  System.out.print("Enter the length from the center to a vertex: ");  radius = scan.nextDouble();  /\*\*  \* Object creation  \*/  Pentagon p1 = new Pentagon(radius);  /\*\*  \* Prints the area according to toString() (Pentagon class)  \*/  System.out.println("The area of the pentagon is " + p1.toString());  }  }  Output  Enter the length from the center to a vertex: 8.8  The area of the pentagon is 184.12 |

1. (5 Points) The great circle distance is the distance between two points on the surface of a sphere. Let (x1, y1) and (x2, y2) be the geographical latitude and longitude of two points. The great circle distance between the two points can be computed using the following formula:

Write a program that prompts the user to enter the latitude and longitude of two points on the earth in degrees and displays its great circle distance. The average earth radius is 6,371.01 km. Note that you need to convert the degrees into radians using the **Math.toRadians** method since the Java trigonometric methods use radians. The latitude and longitude degrees in the formula are for north and west. Use negative to indicate south and east degrees. After executing the program, what did you notice with Radians and degrees?

|  |
| --- |
| package ll;  import java.util.\*;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class Latitude {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  double x1;  double x2;  double y1;  double y2;  /\*\*  \*  \* Reads input from user  \*/  Scanner scan = new Scanner(System.in);  System.out.println("Enter point 1 (latitude and longitude) in degrees: ");  x1 = scan.nextDouble();  y1 = scan.nextDouble();  /\*\*  \*  \* Reads input from user  \*/  System.out.println("Enter point 2 (latitude and longitude) in degrees: ");  x2 = scan.nextDouble();  y2 = scan.nextDouble();  double distance;  /\*\*  \*  \* calculates the distance  \*/  distance = 6371.01 \* Math.acos(Math.sin(Math.toRadians(x1))  \* Math.sin(Math.toRadians(x2))  + (Math.cos(Math.toRadians(x1))  \* Math.cos(Math.toRadians(x2)))  \* Math.cos(Math.toRadians(y1) - Math.toRadians(y2)));  /\*\*  \*  \* Prints the calculated distance  \*/  System.out.println("The distance between the two points is " + distance + " km");  }  }  Output :  Enter point 1 (latitude and longitude) in degrees:  33 -121.89  Enter point 2 (latitude and longitude) in degrees:  54.5 87.37  The distance between the two points is 9889.567927528571 km |

1. (7 Points) Questions on Math class
   1. Evaluate the following method calls.

(a) Math.sqrt(4)

(b) Math.sin(2 \* Math.PI)

(c) Math.cos(2 \* Math.PI)

(d) Math.pow(2, 2)

(e) Math.log(Math.E)

(f) Math.exp(1)

(g) Math.max(2, Math.min(3, 4))

(h) Math.rint(-2.5)

(i) Math.ceil(-2.5)

(j) Math.floor(-2.5)

(k) Math.round(-2.5f)

(l) Math.round(-2.5)

(m) Math.rint(2.5)

(n) Math.ceil(2.5)

(o) Math.floor(2.5)

(p) Math.round(2.5f)

(q) Math.round(2.5)

(r) Math.round(Math.abs(-2.5))

|  |
| --- |
| 2.0  -2.4492935982947064E-16  1.0  4.0  1.0  3  -2.0  -2.0  -3.0  -2  -2  2.0  3.0  2.0  3  3  3 |

8.2 True or false? The argument for trigonometric methods is an angle in radians.

|  |
| --- |
| True |

* 1. Write a statement that converts **47** degrees to radians and assigns the result to a variable.

|  |
| --- |
| double radians = Math.toRadians(47); |

* 1. Write a statement that converts **π/7** to an angle in degrees and assigns the result to a variable

|  |
| --- |
| double degree = Math.toDegrees(Math.PI / 7); |

* 1. Write an expression that obtains a random integer between **34** and **55.** Write an expression that obtains a random integer between **0** and **999.** Write an expression that obtains a random number between **5.5** and **55.5.**

|  |
| --- |
| 34 + (int)(Math.random() \* (55 - 34))  (int)(Math.random() \* 1000)  5.5 + (Math.random() \* (55.5 - 5.5)) |

* 1. Why does the **Math** class not need to be imported?

|  |
| --- |
| Because it's in the java.lang package, which is automatically imported |

* 1. What is **Math.log(Math.exp(5.5))**? What is **Math.exp(Math.log(5.5))**?

What is **Math.asin(Math.sin(Math.PI / 6))**?

What is **Math.sin(Math.asin(Math.PI / 6))**?

|  |
| --- |
| 5.5  5.5  0.5235987755982988  0.5235987755982988 |

1. (8 Points) Java API has the **GregorianCalendar** class in the **java.util** package, which you can use to obtain the year, month, and day of a date. The no-arg constructor constructs an instance for the current date, and the methods **get(GregorianCalendar.YEAR)**, get**(GregorianCalendar.MONTH),**and **get(GregorianCalendar.DAY\_OF\_MONTH)** return the year, month, and day.

Write a program to perform two tasks:

* Display the current year, month, and day.
* The **GregorianCalendar** class has the **setTimeInMillis(long),** which can be used to set a specified elapsed time since January 1, 1970. Set the value to **1234567898765L** and display the year, month, and day.
* Write briefly what did you learn and observe from this program.

|  |
| --- |
| package calendar;  import java.util.GregorianCalendar;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class Calendar {  public static void main(String[] args) {  /\*\*  \*  \* Object Creation  \*/  GregorianCalendar cal = new GregorianCalendar();  System.out.println("Current year:" + cal.get(GregorianCalendar.YEAR));  System.out.println("Current month:" + cal.get(GregorianCalendar.MONTH));  System.out.println("Current day:" + cal.get(GregorianCalendar.DAY\_OF\_MONTH));  cal.setTimeInMillis(1234567898765L);  System.out.println("Updated year:" + cal.get(GregorianCalendar.YEAR));  System.out.println("Updated month:" + cal.get(GregorianCalendar.MONTH));  System.out.println("Updated day:" + cal.get(GregorianCalendar.DAY\_OF\_MONTH));  }  }  Output :  Current year:2020  Current month:8  Current day:27  Updated year:2009  Updated month:1  Updated day:13 |

1. (10 Points) Explain the concept of method overloading method with examples.

|  |
| --- |
| Method Overloading is basically having more than one method having the same name but different arguments, either the type or the number of arguments. It is similar to constructor overloading, where a class can have multiple constructors with different arguments.  eg: add(int, int)  add(int, int, int)  add(string, string) |

1. (10 Points) The monthly payment for a given loan pays the principal and the interest. The monthly interest is computed by multiplying the monthly interest rate and the balance (the remaining principal).The principal paid for the month is therefore the monthly payment minus the monthly interest. Write a program that lets the user enter the loan amount, number of years, and interest rate and displays the amortization schedule for the loan.

|  |
| --- |
| Bank java  package bank;  /\*\*  \*  \* @author s540109  \*/  public class Bank {  private double loan;  private double years;  private double annualInterestRate;  private double totalPayment;  double months = years \* 12;  }  Bank Driver  package bank;  import java.util.Scanner;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class BankDriver {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner input = new Scanner(System.in);  // Prompt the user to enter the loan amount,  // the number of years and the annual interest rate  System.out.print("Loan Amount: ");  double loanAmount = input.nextDouble();  System.out.print("Number of Years: ");  int years = input.nextInt();  System.out.print("Annual Interest Rate: ");  double annualRate = input.nextDouble();  double balance = loanAmount;  double principal;  double interest;  // Calculate monthly interest rate  double monthlyRate = annualRate / 1200;  // Calculat montly payment  double monthlyPayment = loanAmount \* monthlyRate \* Math.pow(1 + monthlyRate, years \* 12) / (Math.pow(1 + monthlyRate, years \* 12) - 1);  // Display montly payment  System.out.printf("Monthly Payment: %.2f\n", monthlyPayment);  // Display total payment  System.out.printf("Total Payment: %.2f\n", (monthlyPayment \* 12) \* years);  // Create amortization schedule  System.out.println("Payment# Interest Principal Balance");  int i = 0;  while (i < years \* 12) {  interest = monthlyRate \* balance;  principal = monthlyPayment - interest;  balance = balance - principal;  System.out.printf("%-13d%-13.2f%-13.2f%.2f\n", i + 1, interest,  principal, balance);  i++;  }  }  } |

1. (10 Points) You have just started a sales job in a department store. Your pay consists of a base salary and a commission. The base salary is $5,000. The scheme shown below is used to determine the commission rate.

|  |
| --- |
| package sales;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class sales {  public static void main(String[] args) {  double baseSalary = 5000;  double target = 30000;  double requiredCommission = target - baseSalary;  double commission = 0;  double saleAmount = 0;  /\*\*  \*  \* Calculating commission for minimum sales  \*/  do {  if (saleAmount < 5000) {  commission = 0.08 \* saleAmount;  } else if ((5000 < saleAmount) && (saleAmount < 10000)) {  commission = (.08 \* 5000) + (0.1 \* (saleAmount - 5000));  } else if (saleAmount > 10000) {  commission = (.08 \* 5000) + (0.1 \* 5000) + (0.12 \* (saleAmount - 10000));  }  saleAmount = saleAmount + .01;  } while (commission <= requiredCommission);  /\* \*  \*  Prints minimum sales  \*/  System.out.printf("The minimum sales needed to earn 30,000: %.2f", saleAmount);  }  }  Output :  The minimum sales needed to earn 30,000: 210833.3 |

1. (15 Points) Credit card numbers follow certain patterns. A credit card number must have between 13 and 16 digits. It must start with:

* 4 for Visa cards
* 5 for Master cards
* 37 for American Express cards
* 6 for Discover cards

In 1954, Hans Luhn of IBM proposed an algorithm for validating credit card numbers. The algorithm is useful to determine whether a card number is entered correctly or whether a credit card is scanned correctly by a scanner. Credit card numbers are generated following this validity check, commonly known as the Luhn check or the Mod 10 check, which can be described as follows (for illustration, consider the card number 4388576018402626):

1. Double every second digit from right to left. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number.

4388576018402626

2 \* 2 = 4

2 \* 2 = 4

4 \* 2 = 8

1 \* 2 = 2

6 \* 2 = 12 (1+2 = 3)

5 \* 2 = 10 (1+0 = 1)

8 \* 2 = 16 (1+6 = 7)

4 \* 2 = 8

1. Now add all single-digit numbers from Step 1.

4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37

1. Add all digits in the odd places from right to left in the card number.

6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38

1. Sum the results from Step 2 and Step 3.
2. + 38 = 75
3. If the result from Step 4 is divisible by 10, the card number is valid; otherwise,

It is invalid. For example, the number 4388576018402626 is invalid, but the

number 4388576018410707 is valid.

Write a program that prompts the user to enter a credit card number as a long integer. Display whether the number is valid or invalid. Design your program to use the following methods:

/\*\* Return true if the card number is valid \*/

**public static boolean** isValid(**long** number)

/\*\* Get the result from Step 2 \*/

**public static int** sumOfDoubleEvenPlace(**long** number)

/\*\* Return this number if it is a single digit, otherwise,

\* return the sum of the two digits \*/

**public static int** getDigit(**int** number)

/\*\* Return sum of odd-place digits in number \*/

**public static int** sumOfOddPlace(**long** number)

/\*\* Return true if the digit d is a prefix for number \*/

**public static boolean** prefixMatched(**long** number, **int** d)

/\*\* Return the number of digits in d \*/

**public static int** getSize(**long** d)

/\*\* Return the first k number of digits from number. If the

\* number of digits in number is less than k, return number. \*/

**public static long** getPrefix(**long** number, **int** k)

Here are sample runs of the program: (You may also implement this program by reading the input as a string and processing the string to validate the credit card.)

|  |
| --- |
| Enter a credit card number as a long integer: 4388576018410707  4388576018410707 is valid |

|  |
| --- |
| Enter a credit card number as a long integer: 4388576018402626  4388576018402626 is invalid |

|  |
| --- |
| Credit Card :  package creditCard;  import java.util.Scanner;  /\*\*  \*  \* @author s540109  \*/  public class CreditCard {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  Scanner scan = new Scanner(System.in);  System.out.println("Enter a credit card number as a long integer: ");  long cardNumber = scan.nextLong();  if (isValid(cardNumber)) {  System.out.println(cardNumber + " is valid");  } else {  System.out.println(cardNumber + " is invalid");  }  }  /\*\*  \* Return true if the card number is valid  \* @param number  \* @return  \*/  public static boolean isValid(long number) {  return (getSize(number) >= 13 && getSize(number) <= 16)  && (prefixMatched(number, 4) || prefixMatched(number, 5)  || prefixMatched(number, 37) || prefixMatched(number, 6))  && ((sumOfDoubleEvenPlace(number) + sumOfOddPlace(number)) % 10 == 0);  }  /\*\*  \* Get the result from Step 2  \* @param number  \* @return  \*/  public static int sumOfDoubleEvenPlace(long number) {  String no = String.valueOf(number);  int sum = 0;  for (int i = getSize(number) - 2; i >= 0; i -= 2) {  sum += getDigit(Integer.valueOf(no.charAt(i)) \* 2);  }  return sum;  }  /\*\*  \* Return this number if it is a single digit, otherwise, return the sum of  \* the two digits  \* @param number  \* @return  \*/  public static int getDigit(int number) {  if (number > 9) {  number = number / 10 + number % 10;  }  return number;  }  /\*\*  \* Return sum of odd-place digits in number  \* @param number  \* @return  \*/  public static int sumOfOddPlace(long number) {  String no = String.valueOf(number);  int sum = 0;  for (int i = getSize(number) - 1; i >= 0; i -= 2) {  sum += Integer.valueOf(no.charAt(i));  }  return sum;  }  /\*\*  \* Return true if the digit d is a prefix for number  \* @param number  \* @param d  \* @return  \*/  public static boolean prefixMatched(long number, int d) {  return String.valueOf(number).startsWith(String.valueOf(d));  }  /\*\*  \* Return the number of digits in d  \* @param d  \* @return  \*/  public static int getSize(long d) {  return String.valueOf(d).length();  }  /\*\*  \* Return the first k number of digits from number.If the number of digits  in number is less than k, return number.  \* @param number  \* @param k  \* @return  \*/  public static long getPrefix(long number, int k) {  if (getSize(number) < k) {  return number;  }  return Long.getLong(String.valueOf(number).substring(0, k));  }  } |

1. Write the source code to count the occurrences of **string2** appears in the **string1**. Read two inputs **string1** and **string2** and find the **string2** occurrences in **string1**

|  |
| --- |
| package occurrences;  import java.util.\*;  /\*\*  \*  \* @author Nithya Karepe  \*/  public class occurrences {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  // TODO code application logic here  String str1 = " ";  String str2;  int count = 0;  Scanner scan = new Scanner(System.in);  System.out.print("Enter string 1: ");  str1 = scan.nextLine();  System.out.print("Enter string 2: ");  str2 = scan.nextLine();  for (int i = 0; i < str1.length(); i++) {  if (str1.charAt(i) == str2.charAt(0) && i < str1.length() - str2.length() + 1) {  if (str1.substring(i, i + str2.length()).equals(str2)) {  count++;  }  }  }  System.out.println(count);  }  } |