Q1: RecursiveMethod [10 points]

static void checkEvenOdd(int num) {  
 if (num == 0) {  
 System.*out*.println("The number is even.");  
 } else if (num == 1 || num == -1) {  
 System.*out*.println("The number is odd.");  
 } else if (num > 0) {  
 *checkEvenOdd*(num - 2);  
 } else {  
 *checkEvenOdd*(num + 2);  
 }  
}

Q2: HexEquivalent [5 points]

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class HexEquivalent {  
  
 static char[] *hexNumber* = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};  
  
 //As said by prof on Ed, no need to consider float, negative number  
 static String getHexEquivalent(int num){  
 if (num < 0) {  
 throw new IllegalArgumentException("negative integer");  
 }  
 int intReminder = num % 16;  
 int intQuotient = num - intReminder;  
 char hexReminder = *hexNumber*[intReminder];  
 if (intQuotient == 0){  
 return String.*valueOf*(hexReminder);  
 } else {  
 return *getHexEquivalent*(intQuotient/16)+hexReminder;  
 }  
 }  
  
//*TODO: static String getHexEquivalent(double num){}* public static void main(String[] args) {  
 try (Scanner scanner = new Scanner(System.*in*)) {  
 System.*out*.println("Enter a decimal number (positive integer): ");  
 int input = scanner.nextInt();  
 System.*out*.println("It's hex equivalent: \n" + *getHexEquivalent*(input));  
// System.out.println(Integer.toHexString(input));  
 } catch (InputMismatchException e) {  
 System.*out*.println("InputMismatchException caught: " + e);  
 } catch (IllegalArgumentException e1) {  
 System.*out*.println("IllegalArgumentException caught: " + e1);  
 }  
 }  
}

Q3: DayOfMonth [10 points]

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class DayOfMonth {  
 static String[] *months* = {"January", "February", "March", "April",  
 "May", "June", "July", "August",  
 "September", "October", "November", "December"};  
 static int[] *dom* = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};  
 public static void main(String[] args) {  
 try (Scanner scanner = new Scanner(System.*in*)) {  
 System.*out*.println("Enter an integer between 1 and 12: ");  
 int input = scanner.nextInt();  
 System.*out*.println(*months*[input - 1] + " has " + *dom*[input - 1] + " days. ");  
 } catch (InputMismatchException e) {  
 System.*out*.println("InputMismatchException caught " );  
 } catch (IndexOutOfBoundsException e2) {  
 System.*out*.println("wrong number");  
 }  
 }  
}

Q4: AdditionOfDigits [10 points]

import java.util.InputMismatchException;  
import java.util.Scanner;  
  
public class AdditionOfDigits {  
 static int RecursiveAddition(int input) {  
 int positiveMark = 1;  
 // uncomment this if you want always positive sum of digits, even for negative input  
 // positiveMark = input >= 0 ? 1 : -1;  
 int reminder = input % 10 \* positiveMark;  
 if (reminder == input) {  
 return reminder;  
 } else {  
 int quotient = (input - reminder) / 10;  
 return *RecursiveAddition*(quotient) + reminder;  
 }  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.println("Please enter an integer: ");  
 while (true) {  
 try {  
 System.*out*.println("Addition of all digits of the number is: " + *RecursiveAddition*(scanner.nextInt()));  
 break;  
 } catch (InputMismatchException e) {  
 System.*out*.println("Enter an integer again: ");  
 scanner.next();  
 }  
 }  
 }  
}

Q5: RecursionPower [5 points]

static long calPower(int x, int n) {  
//uncomment try...catch part if in-method exception catch is needed  
// try {  
 if (n < 0) {  
 throw new ArithmeticException("Exponent must be non-negative");  
 }  
 if (x == 0) {  
 if (n == 0) {  
 throw new ArithmeticException("zero to the power of zero");  
 }  
 return 0;  
 }  
 if (x == 1) {  
 return 1;  
 }  
 if (x == -1) {  
 return n % 2 == 0 ? 1 : -1;  
 }  
 if (n == 0) {  
 return 1;  
 }  
 if (n == 1) {  
 return x;  
 }  
 long temp = *calPower*(x, n / 2);  
 if ((temp>0 && temp > Long.*MAX\_VALUE* / temp) ||  
 (temp<0 && temp < Long.*MAX\_VALUE* / temp)) {  
 throw new ArithmeticException("Overflow occurred during calculation");  
 }  
  
 long tempSquared = temp \* temp;  
 if (n % 2 != 0) {  
 if ((x > 0 && tempSquared > Long.*MAX\_VALUE* / x) ||  
 (x < 0 && tempSquared > Long.*MIN\_VALUE* / x)) {  
 throw new ArithmeticException("Overflow occurred during calculation");  
 }  
 }  
 return tempSquared \* (n % 2 != 0 ? x : 1);  
// } catch (ArithmeticException e) {  
// System.out.println("ArithmeticException thrown: " + e.getMessage());  
// return 0;  
// }  
 }