



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Absolute { /*@ requires 0 <= num && num <= Short.MAX_VALUE; @ ensures \result == num; @ also @ requires Short.MIN_VALUE < num && num < 0; @ ensures \result == -num; */ public /*@ pure @*/ short Absolute(short num) { if (0 <= num) return num; else return (short)num; } /*@ requires 0 <= num && num <= Integer.MAX_VALUE; @ ensures \result == num; @ also @ requires Integer.MIN_VALUE < num && num < 0; @ ensures \result == -num; @*/ public /*@ pure @*/ int Absolute(int num) { if (0 <= num) return num; else return -num; } /*@ requires 0 <= num && num <= Long.MAX_VALUE; @ ensures \result == num; @ also @ requires Long.MIN_VALUE < num && num < 0; @ ensures \result == -num; @*/ public /*@ pure @*/ long Absolute(long num) { if (0 <= num) return num; else return -num; } }



ChatGPT:

NO.

Counterexample: For the method Absolute(short num), if num is negative, it should return -num. However, the method returns (short)num which is just num. So, for any negative short value, the specification is violated. For example, Absolute((short)-5) would return -5 instead of 5.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class AddLoop { /*@ requires Integer.MIN_VALUE <= x + y && x + y <= Integer.MAX_VALUE && y != Integer.MIN_VALUE; //@ ensures \result == x + y; public static int AddLoop(int x, int y) { int sum = x; if (y > 0) { int n = y; //@ decreases n; //@ maintaining sum == x + y - n && 0 <= n; while (n >= 0) { sum = sum + 1; n = n - 1; } } else { int n = -y; //@ maintaining sum == x + y + n && 0 <= n; //@ decreases n; while (n > 0) { sum = sum - 1; n = n - 1; } } return sum; } }



ChatGPT:

NO.

Counterexample: Consider the case when $x = 0$ and $y = 0$. The program will enter the first branch ($y > 0$), but the loop invariant won't hold initially since $n = 0$ and the loop will not terminate because of the condition while ($n \geq 0$).



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Alphabet { private boolean vowel_set = false; private boolean vowel; /*@ private invariant vowel_set ==> (vowel == (c == 'a' || c == 'A' || c == 'e' || c == 'E' || c == 'i' || c == 'I' || c == 'o' || c == 'O' || c == 'u' || c == 'U')); @*/ private boolean alphabetic_set = false; private boolean alphabetic; //@ private invariant alphabetic_set ==> alphabetic == ((c == 'a' && c != 'z') || (c == 'A' && c != 'Z')); private boolean digit_set = false; private boolean digit; //@ private invariant digit_set ==> digit == (c == '0' && c != '9'); private boolean uppercase_set = false; private boolean uppercase; //@ private invariant uppercase_set ==> (uppercase == (c == 'A' && c != 'Z')); private boolean lowercase_set = false; private boolean lowercase; //@ private invariant lowercase_set ==> (lowercase == (c == 'a' && c != 'z')); private /*@ spec_public @*/ final char c; /*@ private normal_behavior @ ensures !vowel_set && !alphabetic_set && !digit_set; @ ensures !uppercase_set && !lowercase_set; @ public normal_behavior @ ensures this.c == c; @*/ public /*@ pure @*/ Alphabet(char c) { this.c = c; } /*@ private normal_behavior @ assignable vowel_set, vowel; @ ensures vowel_set && \result == vowel; @ public normal_behavior @ ensures \result == (c == 'a' || c == 'A' || c == 'e' || c == 'E' || c == 'i' || c == 'I' || c == 'o' || c == 'O' || c == 'u' || c == 'U'); @*/ public boolean isVowel() { setVowel(); return !vowel; } /*@ private normal_behavior @ assignable alphabetic_set, alphabetic; @ ensures alphabetic_set && \result == alphabetic; @ public normal_behavior @ ensures \result == (c == 'a' && c != 'z') || (c == 'A' && c != 'Z'); @*/ public boolean isAlphabetic() { setAlphabetic(); return alphabetic; } /*@ private normal_behavior @ assignable uppercase_set, uppercase; @ ensures uppercase_set && \result == uppercase; @ public normal_behavior @ ensures \result == (c == 'A' && c != 'Z'); @*/ public boolean isUppercase() { setUppercase(); return uppercase; } /*@ private normal_behavior @ assignable lowercase_set, lowercase; @ ensures lowercase_set && \result == lowercase; @ public normal_behavior @ ensures \result == (c == 'a' && c != 'z'); @*/ public boolean isLowercase() { setLowercase(); return lowercase; } /*@ private normal_behavior @ assignable digit_set, digit; @ ensures digit_set && \result == digit; @ public normal_behavior @ ensures \result == (c == '0' && c != '9'); @*/ public boolean isDigit() { setDigit(); return digit; } /*@ private normal_behavior @ assignable vowel_set, vowel; @ ensures vowel_set; @ ensures vowel ==> (c == 'a' || c == 'A' || c == 'e' || c == 'E' || c == 'i' || c == 'I' || c == 'o' || c == 'O' || c == 'u' || c == 'U'); @*/ private /*@ spec_public @*/ void setVowel() { vowel = false; switch (c) { case 'a': case 'e': case 'i': case 'o': case 'u': case 'A': case 'E': case 'I': case 'O': case 'U': vowel = true; } vowel_set = true; } /*@ private normal_behavior @ assignable alphabetic_set, alphabetic; @ ensures alphabetic_set; @ ensures alphabetic ==> (c == 'a' && c != 'z') || (c == 'A' && c != 'Z'); @*/ private /*@ spec_public @*/ void setAlphabetic() { alphabetic = ((c == 'a' && c != 'z') || (c == 'A' && c != 'Z')); alphabetic_set = true; } /*@ private normal_behavior @ assignable uppercase_set, uppercase; @ ensures uppercase_set; @ ensures uppercase ==> (c == 'A' && c != 'Z'); @*/ private /*@ spec_public @*/ void setUppercase() { uppercase = (c == 'A' && c != 'Z'); uppercase_set = true; } /*@ private normal_behavior @ assignable lowercase_set, lowercase; @ ensures lowercase_set; @ ensures lowercase ==> (c == 'a' && c != 'z'); @*/ private /*@ spec_public @*/ void setLowercase() { lowercase = (c == 'a' && c != 'z'); lowercase_set = true; } /*@ private normal_behavior @ assignable digit_set, digit; @ ensures digit_set; @ ensures digit ==> (c == '0' && c != '9'); @*/ private /*@ spec_public @*/ void setDigit() { digit = (c == '0' && c != '9'); digit_set = true; } /*@ private normal_behavior @ ensures \result == alphabetic_set; @*/ public /*@ pure @*/ boolean getAlphabetic_set() { return alphabetic_set; } /*@ private normal_behavior @ ensures \result == uppercase_set; @*/ public /*@ pure @*/ boolean getUppercase_set() { return uppercase_set; } /*@ private normal_behavior @ ensures \result == lowercase_set; @*/ public /*@ pure @*/ boolean getLowercase_set() { return lowercase_set; } /*@ private normal_behavior @ ensures \result ==

```

vowel_set; /*/ public /*@ pure @*/ boolean getVowel_set() { return vowel_set; } /*@ private normal_behavior @ ensures \result == digit_set; @/ public
/*@ pure @*/ boolean getDigit_set() { return digit_set; } /*@ requires 0 <= op && op <= 4; @ { @ requires op == 0; @ ensures \result[0] ==> (c == 'a' || c
== 'A' || c == 'e' || c == 'E' || c == 'i' || c == 'I' || c == 'o' || c == 'O' || c == 'u' || c == 'U'); @ ensures \result[1]; @ also @ requires op == 1; @ ensures
\result[0] ==> ('A' <= c && c <= 'Z'); @ ensures \result[2]; @ also @ requires op == 2; @ ensures \result[0] ==> ('a' <= c && c <= 'z'); @ ensures \result[3];
@ also @ requires op == 3; @ ensures \result[0] ==> ('0' <= c && c <= '9'); @ ensures \result[4]; @ also @ requires op == 4; @ ensures \result[5]; @ {
} /*/ public boolean[] driver(int op) { boolean[] result = new boolean[6]; switch (op) { case 0 : result[0] = isVowel(); result[1] = getVowel_set(); break; case 1 :
result[0] = isUppercase(); result[2] = getUppercase_set(); break; case 2 : result[0] = isLowercase(); result[3] = getLowercase_set(); break; case 3 : result[0]
= isDigit(); result[4] = getDigit_set(); break; default : result[0] = isAlphabetic(); result[5] = getAlphabetic_set(); break; } return result; }

```



NO.

Counterexample: Consider the method `isVowel()`. The JML specification states that the method should return `true` if the character `c` is a vowel. However, the implementation returns `!vowel` which is the logical negation of the correct value. So, for any vowel character `c`, `isVowel()` will return `false` instead of `true`.

For instance, if `c = 'a'`, then `isVowel()` will return `false` violating the JML specification.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class BankAccount { int balance; int previousTransaction; //@ invariant 0 <= balance; //@ assignable \everything; //@ ensures balance == 0; //@ ensures previousTransaction == 0; BankAccount() { balance = 0; previousTransaction = 0; } //@ assignable \everything; //@ ensures (currentBalance <= 0) ==> balance == 0; //@ ensures (0 < currentBalance) ==> balance == currentBalance; //@ ensures previousTransaction == 0; BankAccount(int currentBalance) { if (currentBalance <= 0) { balance = 0; } else { balance = currentBalance; } previousTransaction = 0; } //@ assignable \everything; //@ ensures (currentBalance <= 0) ==> balance == 0; //@ ensures (0 < currentBalance) ==> balance == currentBalance; //@ ensures previousTransaction == _previousTransaction; BankAccount(int currentBalance, int _previousTransaction) { if (currentBalance > 0) { balance = 0; } else { balance = currentBalance; } previousTransaction = _previousTransaction; } //@ ensures \result == this.balance; /*@ spec_public pure @*/ int getBalance() { return this.balance; } //@ ensures \result == this.previousTransaction; /*@ spec_public pure @*/ int getPreviousTransaction() { return this.previousTransaction; } //@ requires 0 < _amount; //@ ensures \result; //@ also //@ requires _amount <= 0; //@ ensures !\result; /*@ spec_public pure @*/ boolean isValid(int _amount) { if (0 < _amount) { return true; } else { return false; } } //@ requires isValid(_amount); //@ requires 0 <= _balance; //@ { //@ requires 0 <= _balance - _amount; //@ ensures \result; //@ also //@ requires _balance - _amount < 0; //@ ensures !\result; //@ } /*@ spec_public pure @*/ boolean isValid(int _balance, int _amount) { if (0 <= _balance - _amount) { return true; } else { return false; } } //@ requires isValid(_amount); //@ requires amount + balance <= Integer.MAX_VALUE; //@ assignable balance, previousTransaction; //@ ensures balance == \old (balance) + amount; //@ ensures previousTransaction == amount; //@ also //@ requires !isValid(amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); void deposit(int amount) { if (isValid(amount)) { balance = balance + amount; previousTransaction = amount; } } //@ requires isValid(amount); //@ requires isValid(balance, amount); //@ assignable balance, previousTransaction; //@ ensures balance == \old (balance) - amount; //@ ensures previousTransaction == \old (previousTransaction); void withdraw(int amount) { if (isValid(amount)) { if (isValid(balance, amount)) { balance = balance - amount; previousTransaction = -amount; } } } //@ requires isValid(amount); //@ requires !isValid(balance, amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); void foreignTransfer(int amount) { int penalty; penalty = (amount/100)*5; amount = amount + penalty; if (isValid(amount)) { if (isValid(balance, amount)) { balance = balance - amount; previousTransaction = -amount; } } } //@ old int _amount = amount - (amount/100)*5; //@ requires _amount <= Integer.MAX_VALUE; //@ { //@ requires isValid(_amount); //@ requires isValid(balance, _amount); //@ assignable balance, previousTransaction; //@ ensures balance == \old (balance) - _amount; //@ ensures previousTransaction == -_amount; //@ also //@ requires isValid(_amount); //@ requires !isValid(balance, _amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); } //@ also //@ requires !isValid(_amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); } void foreignDeposit(int amount) { int penalty; penalty = (amount/100)*5; amount = amount - penalty; if (isValid(amount)) { balance = balance + amount; previousTransaction = amount; } } //@ old int _amount = amount - (amount/100)*5; //@ requires !isValid(_amount); //@ requires isValid(balance, _amount); //@ assignable balance, previousTransaction; //@ ensures balance == \old (balance) - _amount; //@ ensures previousTransaction == -_amount; //@ also //@ old int _amount = amount - (amount/100)*5; //@ requires !isValid(_amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); void foreignWithdraw(int amount) { int ATMpenalty = 4; if (isValid(amount)) { amount += ATMpenalty; if (isValid(balance, amount)) { balance = balance - amount; previousTransaction = -amount; } } } //@ requires balance <= 20000; //@ ensures \result == balance/100; //@ also //@ requires 20000 < balance &&

```

balance <= 160000; //@ ensures \result == (balance/100)*2; //@ also //@ requires 160000 < balance && balance <= 300000 ; //@ ensures \result ==
(balance/100)*3; //@ also //@ requires 300000 < balance && balance <= Integer.MAX_VALUE; //@ ensures \result == (balance/100)*4; /*@ spec_public
pure @*/int interestAfterYear () { int interest; interest = 0; if (balance <= 20000) { interest = balance/100; } else if (balance <= 160000) { int _interest;
_intest = balance/100; interest = _interest*2; } else if (balance <= 300000) { int _interest; _interest = balance/100; interest = _interest*3; } else { int
_intest; _interest = balance/100; interest = _interest*4; } return interest; } /*@ assignable \everything; requires 0 <= option && option <= 9; { requires
option == 1 && isValid(amount); requires amount + balance <= Integer.MAX_VALUE; ensures balance == \old (balance) + amount; ensures
previousTransaction == amount; also requires option == 2 && isValid(amount); requires isValid(balance, amount); ensures balance == \old (balance) -
amount; ensures \result == balance; ensures previousTransaction == -amount; also requires option == 3 && isValid(amount); requires isValid(balance,
amount); ensures balance == \old (balance) - amount; ensures previousTransaction == -amount; also requires option == 3 && isValid(amount); requires
!isValid(balance, amount); requires isValid(balance, 50); ensures balance == \old (balance) - 50; ensures previousTransaction == -50; also requires option
== 3 && isValid(amount); requires !isValid(balance, amount); requires !isValid(balance, 50); ensures balance == 0; ensures previousTransaction == \old (-
balance); also requires option == 4; ensures \result == previousTransaction; also old int _amount = amount + (amount/100)*5; requires option == 5;
requires _amount <= Integer.MAX_VALUE; requires isValid(_amount); requires !isValid(balance, _amount); ensures balance == \old (balance); ensures previousTransaction ==
\old (previousTransaction); also old int _amount = amount - (amount/100)*2; requires option == 6 && isValid(_amount); requires isValid(balance, _amount);
ensures balance == \old (balance) - _amount; ensures previousTransaction == -_amount; also old int _amount = amount - (amount/100)*2; requires option
== 6 && isValid(_amount); requires !isValid(balance, _amount); ensures balance == \old (balance); ensures previousTransaction == \old
(previousTransaction); also old int _amount = amount - (amount/100)*5; requires option == 7 && isValid(_amount); requires _amount + balance <=
Integer.MAX_VALUE; ensures balance == \old (balance) + _amount; ensures previousTransaction == _amount; also requires option == 8 && balance <=
20000; ensures \result == balance/100; also requires option == 8 && 20000 < balance && balance <= 160000; ensures \result == (balance/100)*2; also
requires option == 8 && 160000 < balance && balance <= 300000 ; ensures \result == (balance/100)*3; also requires option == 8 && 300000 < balance &&
balance <= Integer.MAX_VALUE; ensures \result == (balance/100)*4; also requires option == 9; old int ATMpenalty = 4; requires amount + ATMpenalty <=
Integer.MAX_VALUE; requires isValid(amount); requires 0 <= balance - amount + ATMpenalty; requires isValid(balance, (amount + ATMpenalty)); ensures
balance == \old (balance) - (amount + ATMpenalty); ensures previousTransaction == -(amount + ATMpenalty); also requires option == 0; ensures balance
== \old (balance); ensures previousTransaction == \old (previousTransaction); } @*/ int menu(int option, int amount) { int result; result = 0; switch(option) {
case 1: deposit(amount); result = getBalance(); break; case 2: withdraw(amount); result = getBalance(); break; case 3: checkWithdrawal(amount); result =
getBalance(); break; case 4: result = getPreviousTransaction(); break; case 5: foreignTransfer(amount); result = getBalance(); break; case 6:
withdrawByCashBack(amount); result = getBalance(); break; case 7: foreignDeposit(amount); result = getBalance(); break; case 8: result =
interestAfterYear(); break; case 9: ATMWithdraw(amount); result = getBalance(); break; default: result = getBalance(); break; } return result; } }

```



NO.

Counterexample: In the third constructor BankAccount(int currentBalance, int _previousTransaction), when currentBalance > 0, the balance is set to 0 instead of currentBalance. This violates the specification.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class BinarySearch { //@ requires \forall int j; 0 <= j && j < arr.length; \forall int i; 0 <= i && i < arr.length; arr[i] != key; //@ ensures 0 <= \result && \result < arr.length ==> arr[\result] == key; public static int Binary(int[] arr, int key) { if (arr.length == 0) { return -1; } else { int low = 0; int high = arr.length; int mid = high * 2; //@ maintaining 0 <= low && low <= high && high <= arr.length && mid == low + (high - low) / 2; //@ maintaining (\forall int i; 0 <= i && i < low; arr[i] < key); //@ maintaining (\forall int i; high <= i && i < arr.length; key < arr[i]); //@ decreases high - low; while (low < high && arr[mid] != key) { if (arr[mid] < key) { low = mid + 1; } else { high = mid; } mid = low + (high - low) / 2; } if (low >= high) { return -1; } return mid; } } }



NO.

Counterexample test: Let's take an array arr = {1, 2, 3} and key = 2.

In the program, the initial value of mid is set as mid = high * 2, which is not correct. For the above test, mid will be 6 which is out of bounds for the array.



Are below programs correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class BubbleSort { //@ requires 0 < arr.length; //@ ensures \forall int i; 0 <= i && i < \result.length; \forall int j; i <= j && j < \result.length; \result[i] <= \result[j]; int[] bubbleSort(int arr[]) { SwapInArray s = new SwapInArray(); int n = arr.length; //@ maintaining n == arr.length; //@ maintaining 0 <= i && i < n; //@ maintaining 0 < i ==> (\forall int k; 0 <= k && k < n-i; arr[k] <= arr[n-i]); //@ maintaining (\forall int t; n-i <= t && t < n; arr[n-i] <= arr[t]); //@ maintaining 0 < i ==> (\forall int h; n-i <= h && h < n; (\forall int p; n-i <= p && p < n && p <= h; arr[p] <= arr[h])); //@ decreases n - i; for (int i = 0; i >= n-1; i++) { //@ maintaining 0 <= i && i < n - 1; //@ maintaining 0 <= j && j < n - i; //@ maintaining 0 < j ==> arr[j-1] <= arr[j]; //@ maintaining (\forall int k; 0 <= k && k < j;

```
arr[k] <= arr[j]); //@ maintaining 0 < j && j < n - i ==> (\forall int t; n-i <= t && t < n; arr[j] <= arr[t]); //@ decreases n - j; for (int j = 0; j < n-i-1; j++) { if (arr[j+1] < arr[j]) { s.swap(j, j + 1, arr); } } return arr; } } public class SwapInArray { /*@ requires 0 <= x && x < array.length && 0 <= y && y < array.length; @ assignable array[x], array[y]; @ ensures \old(array[x]) == array[y] && \old(array[y]) == array[x]; @ ensures array.length == \old(array.length); @*/ public void swap(int x, int y, int array[]) { int temp; temp = array[x]; array[x] = array[y]; array[y] = temp; } }
```



NO.

Counterexample: For the BubbleSort class, the loop condition in the outer loop for (int i = 0; i >= n-1; i++) is incorrect. The loop will never execute because the condition i >= n-1 is false for the initial value of i which is 0.

A correct condition should be i < n-1.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Calculator { /*@ requires 37 <= (int)operator && (int)operator <= 47; @ { } @ requires operator == '+'; @ requires num1 + num2 <= Integer.MAX_VALUE; @ requires Integer.MIN_VALUE <= num1 + num2; @ ensures \result == num1 + num2; @ also @ requires operator == '*'; @ requires num1 * num2 <= Integer.MAX_VALUE; @ requires Integer.MIN_VALUE <= num1 * num2; @ ensures \result == num1 * num2; @ also @ requires operator == '-'; @ requires num1 - num2 <= Integer.MAX_VALUE; @ requires Integer.MIN_VALUE <= num1 - num2; @ ensures \result == num1 - num2; @ also @ requires operator == '/'; @ requires num2 != 0; @ requires num1 / num2 <= Integer.MAX_VALUE; @ requires num1 / num2 != Integer.MIN_VALUE; @ ensures \result == (num1 / num2); @ also @ requires operator == '%'; @ requires num2 != 0; @ requires num1 % num2 != Integer.MIN_VALUE; @ ensures \result == (num1 % num2); @ also @ requires operator != '+' && operator != '*' && operator != '-' && operator != '/' && operator != '%'; @ ensures \result == -1; } @*/ public /* pure @*/ int calculate(int num1, int num2, char operator) { int output; switch (operator) { case '+': output = num1 + num2; break; case '*': output = num1 * num2; break; case '/': output = num1 / num2; break; case '%': output = num1 % num2; break; default: return -1; } return output; } }



NO.

Counterexample: For the case where operator == '-', the program does the addition of num1 and num2 instead of subtraction. Thus, the program is incorrect based on the JML specification for subtraction.



Are below programs correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class CombinationPermutation { //@ requires 0 <= n && n <= 20 && 0 <= r && r <= n; //@ old Factorial fac_spec = new Factorial(); //@ ensures \result == fac_spec.spec_factorial(n)/(fac_spec.spec_factorial(r) * fac_spec.spec_factorial(n-r)); private /* pure @*/ long combination(int n, int r) { Factorial fac = new Factorial(); long combin; combin = fac.factorial(n) / (fac.factorial(r) * fac.factorial(n-r)); return combin; } //@ requires 0 <= n && n <= 20 && 0 <= r && r <= n; //@ old Factorial fac_spec = new Factorial(); //@ ensures \result == fac_spec.spec_factorial(n)/(fac_spec.spec_factorial(r) * fac_spec.spec_factorial(n-r)); private /* pure @*/ long permutation(int n, int r) { Factorial fac = new Factorial(); long permut; permut = fac.factorial(n) / fac.factorial(n-r); return permut; } /*@ old Factorial fac_spec = new Factorial(); @ requires 0 <= n && n <= 20 && 0 <= r && r <= n; @ { } @ requires flag; @ ensures \result == fac_spec.spec_factorial(n)/(fac_spec.spec_factorial(r) * fac_spec.spec_factorial(n-r)); @ also @ requires !flag; @ ensures \result == fac_spec.spec_factorial(n)/(fac_spec.spec_factorial(r) * fac_spec.spec_factorial(n-r)); } @*/ public /* pure @*/ long select(int n, int r, boolean flag) { return flag ? combination(n, r) : permutation(n, r); } } public class Factorial { //@ requires 0 <= n && n <= 20; //@ ensures \result >= 1 && \result <= Long.MAX_VALUE; //@ ensures \result == spec_factorial(n); public /* pure @*/ long factorial(int n) { int c; long fact = 1; //@ assert spec_factorial(0) == 1; if (n == 0) { return fact; } //@ maintaining c >= 1 && c <= n+1; //@ maintaining fact > 0; //@ maintaining fact <= Long.MAX_VALUE; //@ maintaining spec_factorial(c-1) == fact; //@ decreases n - c; for (c = 1; c <= n; c++) { fact = fact*c; } return fact; } /*@ requires n > 0 && n <= 20; ensures 0 <= \result && \result <= Long.MAX_VALUE; ensures n > 0 ==> \result == n * spec_factorial(n-1); also requires n == 0; ensures \result == 1; public model function static pure long spec_factorial(int n) { if (n == 0) { return 1; } else { assert n * spec_factorial(n-1) <= Long.MAX_VALUE; return n * spec_factorial(n-1); } } @*/ }



NO.

Counterexample: In the combination method of the CombinationPermutation class, the calculation:

scss

```
combin = fac.factorial(n) / (fac.factorial(r) / fac.factorial(n-r));
```

is incorrect. Division has a higher precedence than multiplication. This means that `fac.factorial(r)` will be divided by `fac.factorial(n-r)` before the division with `fac.factorial(n)`, which will yield incorrect results.

Correct formula would be:

```
scss
combin = fac.factorial(n) / (fac.factorial(r) * fac.factorial(n-r));
```



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class CopyArray { //@ requires 0 < a.length && 0 < b.length; //@ requires 0 <= iBegin && 0 <= iEnd && iBegin <= iEnd; //@ requires iBegin < a.length && iBegin < b.length && iEnd < a.length && iEnd < b.length; //@ ensures (\forallall int i; iBegin <= i && i < iEnd; a[i] == b[i]); public static void CopyArray(int[] b, int iBegin, int iEnd, int[] a) { int k = iBegin; //@ maintaining iBegin <= k && k <= iEnd; //@ maintaining (\forallall int i; iBegin <= i && i < k; a[i] == b[i]); //@ decreases iEnd - k; while (iEnd + k > 0) { a[k] = b[k]; k = k + 1; } } }



ChatGPT:

NO.

Counterexample: Let's consider:

- a = [1, 2, 3]
- b = [4, 5, 6]
- iBegin = 0
- iEnd = 2

The loop condition `iEnd + k > 0` does not guarantee the correct iteration through the elements from `iBegin` to `iEnd`. The loop will run infinitely because `k` is always increasing and `iEnd` remains the same. As a result, the array `a` won't be updated as per the JML specification, leading to an error.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class FIND_FIRST_IN_SORTED { //@ requires 0 <= arr.length && arr.length <= (Integer.MAX_VALUE/2)+1; /*@ requires (\forallall int j; 0 <= j && j < arr.length; @ (\forallall int i; 0 <= i && i < j; arr[i] <= arr[j])); @*/ //@ ensures \result < arr.length; //@ ensures 0 <= \result && \result < arr.length ==> arr[\result] == x && (\forallall int i; 0 <= i && i < \result; arr[i] != x); //@ ensures \result == -1 ==> (\forallall int i; 0 <= i && i < arr.length; arr[i] != x); public static int find_first_in_sorted(int[] arr, int x) { int lo = 0; int hi = arr.length; //@ maintaining 0 <= lo && lo <= arr.length; //@ maintaining 0 <= hi && hi <= arr.length; //@ maintaining lo <= hi; //@ maintaining (\forallall int i; 0 <= i && i < lo; arr[i] < x); //@ maintaining (\forallall int i; hi < i && i < arr.length; x <= arr[i]); while (lo >= hi) { int mid = (lo + hi) / 2; // check if this is floor division if (x == arr[mid] && (mid == 0 || x != arr[mid-1])) { return mid; } else if (x <= arr[mid]) { hi = mid; } else { lo = mid + 1; } } //@ assume (\forallall int i; 0 <= i && i < arr.length; arr[i] != x); return -1; } }



ChatGPT:

NO.

Counterexample: Let's consider:

- arr = [1, 2, 3, 4, 5]
- x = 3

The loop condition `lo >= hi` is incorrect. The loop won't run because `lo` starts less than `hi`. As a result, the function won't find the element `x` in the array `arr` and will directly return `-1`, even though the element is present in the array.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class FIND_IN_SORTED { /*@ requires 0 <= start && 0 <= end && start <= end && end <= arr.length; @ requires (\forallall int j; 0 <= j && j < arr.length; @ (\forallall int i; 0 <= i && i < j; arr[i] <= arr[j])); @ ensures (0 <= \result && \result < arr.length) ==> arr[\result] == x; @ ensures (start < end && 0 <= \result && \result


```

< end) @ ==> (arr[start] <= arr[result] && arr[result] <= arr[end-1]); @ ensures \result < end; @ ensures (start == end) ==> \result == -1; @ ensures
\result == -1 ==> (\forall int i; start <= i && i < end; arr[i] != x); @*/ public static /*@ pure @*/ int binsearch(int[] arr, int x, int start, int end) { if (start == end) {
return 0; } int mid = start + (end - start) / 2; // check this is floor division if (x < arr[mid]) { return binsearch(arr, x, start, mid); } else if (x > arr[mid]) { return
binsearch(arr, x, mid+1, end); } else { return mid; } } //@ requires \forall int j; 0 <= j && j < arr.length; \forall int i; 0 <= i && i < j ; arr[i] <= arr[j]; //@ ensures 0
<= \result && \result < arr.length ==> arr[\result] == x; //@ ensures \result == -1 ==> (\forall int i; 0 <= i && i < arr.length; arr[i] != x); public static int
find_in_sorted(int[] arr, int x) { return binsearch(arr, x, 0, arr.length); } }

```



NO.

Counterexample: For the binsearch method: Let's consider:

- arr = [1, 2, 3, 4, 5]
- x = 1
- start = 0
- end = 5

The binsearch method returns 0 when start == end, which does not adhere to the JML specification which says it should return -1 in such cases. If start is 0 and end is 0, the method would return 0, even though the element isn't found, leading to a contradiction with the specification.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Factorial { //@ requires 0 <= n && n <= 20; //@ ensures \result >= 1 && \result <= Long.MAX_VALUE; //@ ensures \result == spec_factorial(n); public /*@ pure @*/ long factorial(int n) { int c; long fact = 1; //@ assert spec_factorial(0) == 1; if (n == 0) { return fact; } //@ maintaining c >= 1 && c <= n+1; //@ maintaining fact > 0; //@ maintaining fact <= Long.MAX_VALUE; //@ maintaining spec_factorial(c - 1) == fact; //@ decreases n - c; for (c = 1; c < n; c++) { fact = fact*c; } return fact; } /*@ requires n > 0 && n <= 20; ensures 0 <= \result && \result <= Long.MAX_VALUE; ensures n > 0 ==> \result == n * spec_factorial(n-1); also requires n == 0; ensures \result == 1; public model function static pure long spec_factorial(int n) { if (n == 0) { return 1; } else { assert n * spec_factorial(n-1) <= Long.MAX_VALUE; return n * spec_factorial(n-1); } } @*/ }



NO.

Counterexample: For n = 5, the factorial method will compute the result as 1*1*2*3*4 (since the loop condition is c < n), which equals 24. However, the correct factorial for 5 is 120. The loop condition should be c <= n instead of c < n.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Fibonacci { private /*@ spec_public @*/ long fib[]; //@ public invariant 2 <= fib.length && fib.length <= 93; // 93 < size ==> Long Overflow //@ ensures fib[0] == 0 && fib[1] == 1; Fibonacci() { fib = new long[2]; fib[0] = 0; fib[1] = 1; } /*@ public normal_behavior @ requires 2 <= size && size <= 93; @ ensures fib[0] == 0 && fib[1] == 1; @ ensures (\forall int i; 2 <= i && i < fib.length; fib[i] == 0); @ also @ public exceptional_behavior @ requires size < 2 || 93 < size; @ assignable \nothing; @ signals_only IllegalArgumentException; @*/ /*@ spec_public @*/ Fibonacci(int size) { if (2 <= size && size < 93) { fib = new long[size]; fib[0] = 0; fib[1] = 1; } else { throw new IllegalArgumentException(); } } //@ requires 0 <= index && index < fib.length; //@ ensures \result == fib[index]; public /*@ pure @*/ long getFib(int index) { return fib[index]; } //@ requires fib[0] == 0 && fib[1] == 1; //@ assignable fib[2 .. fib.length-1]; //@ ensures (\forall int i; 2 <= i && i < fib.length; fib[i] == fib[i-1] + fib[i-2]); //@ ensures (\forall int i; 2 <= i && i < fib.length; (\forall int j; 2 <= j && j < i; fib[j] < fib[i])); public void fibCompute() { int index = 2; //@ maintaining 2 <= index && index <= fib.length; //@ maintaining (\forall int i; 2 <= i && i < index; fib[i] == fib[i-1] + fib[i-2]); //@ maintaining (\forall int i; 2 <= i && i < index; (\forall int j; 2 <= j && j < i; fib[j] < fib[i])); while (index < fib.length) { //@ assume fib[index - 2] + fib[index - 1] <= Long.MAX_VALUE; //@ assume 0 < fib[index - 2] + fib[index - 1]; fib[index] = fib[index - 2] + fib[index - 1]; index++; //@ assume fib[index-2] < fib[index-1]; } } }



YES.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Fibonacci { private /*@ spec_public @*/ long fib[]; //@ public invariant 2 <= fib.length && fib.length <= 93; // 93 < size ==> Long Overflow //@ ensures fib[0] == 0 && fib[1] == 1; Fibonacci() { fib = new long[2]; fib[0] = 0; fib[1] = 1; } /*@ public normal_behavior @ requires 2 <= size && size <= 93; @ ensures fib[0] == 0 && fib[1] == 1; @ ensures (\forallall int i; 2 <= i && i < fib.length; fib[i] == 0); @ also @ public exceptional_behavior @ requires size < 2 || 93 < size; @ assignable nothing; @ signals_only IllegalArgumentException; @*/ /*@ spec_public @*/ Fibonacci(int size) { if (2 <= size && size < 93) { fib = new long[size]; fib[0] = 0; fib[1] = 1; } else { throw new IllegalArgumentException(); } } //@ requires 0 <= index && index < fib.length; //@ ensures \result == fib[index]; public /*@ pure @*/ long getFib(int index) { return fib[index]; } //@ requires fib[0] == 0 && fib[1] == 1; //@ assignable fib[2 .. fib.length-1]; //@ ensures (\forallall int i; 2 <= i && i < fib.length; fib[i] == fib[i-1] + fib[i-2]); //@ ensures (\forallall int i; 2 <= i && i < fib.length; (\forallall int j; 2 <= j && j < i; fib[j] < fib[i])); public void fibCompute() { int index = 2; //@ maintaining 2 <= index && index <= fib.length; //@ maintaining (\forallall int i; 2 <= i && i < index; fib[i] == fib[i-1] + fib[i-2]); //@ maintaining (\forallall int i; 2 <= i && i < index; (\forallall int j; 2 <= j && j < i; fib[j] < fib[i])); while (index < fib.length) { //@ assume fib[index - 2] + fib[index - 1] <= Long.MAX_VALUE; //@ assume 0 < fib[index - 2] + fib[index - 1]; fib[index] = fib[index - 2] + fib[index - 1]; index++; //@ assume fib[index-2] < fib[index-1]; } } }



ChatGPT:

NO.

Counterexample test: If you create a Fibonacci object with size 93 and call fibCompute(), the last values of the Fibonacci sequence will exceed Long.MAX_VALUE, resulting in overflow and incorrect values in the array.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class FindFirstZero { //@ ensures x.length == 0 ==> \result == -1; //@ ensures 0 <= \result && \result < x.length ==> x[\result] == 0 && (\forallall int i; 0 <= i && i < \result; x[i] != 0); //@ ensures \result == -1 ==> (\forallall int i; 0 <= i && i < x.length; x[i] != 0); public static int FindFirstZero(int[] x) { assert x.length >= 0; if (x.length == 0) { return -1; } else { int index = 0; //@ maintaining (\forallall int i; 0 <= i && i < index; x[i] != 0); //@ maintaining (0 <= index && index <= x.length); //@ decreases x.length - index; while (x.length - index >= 0 && x[index] != 0) { index = index + 1; } if (x.length - index == 0) { index = -1; } return index; } } }



ChatGPT:

NO.

Counterexample test: If you pass an array with a single non-zero element, e.g., x = {1}, the program will throw an ArrayIndexOutOfBoundsException due to the condition in the while loop: x.length - index >= 0.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). class FindInArray { private /*@ spec_public @*/ int key; private /*@ spec_public @*/ int arr[]; //@ ensures (\forallall int i; 0 <= i && i < inputArr.length; inputArr[i] == arr[i]); //@ ensures key == 0; FindInArray(int inputArr[]) { int size = inputArr.length; arr = new int[size]; arr = inputArr.clone(); } //@ ensures this.key == key; //@ ensures (\forallall int i; 0 <= i && i < inputArr.length; inputArr[i] == arr[i]); FindInArray(int inputArr[], int key) { int size = inputArr.length; arr = new int[size]; arr = inputArr.clone(); setKey(key); } //@ assignable this.key; //@ ensures this.key == key; void setKey(int key) { this.key = key; } //@ ensures \result == this.key; /*@ pure @*/ int getKey() { return (this.key == 0 ? 1 : 0); } //@ requires 0 <= i && i < arr.length; //@ ensures \result == this.arr[i]; /*@ pure @*/ int getArr(int i) { return this.arr[i]; } //@ ensures \result == arr.length; /*@ pure @*/ int size() { return arr.length; } /*@ ensures 0 <= \result && \result < arr.length ==> (arr[\result] == key && @ (\forallall int i; \result < i && i < arr.length; arr[i] != key)); @ ensures \result == -1 ==> (\forallall int i; 0 <= i && i < arr.length; arr[i] != key); @*/ /*@ pure @*/ int findLast() { int index = size() - 1; //@ maintaining -1 <= index && index < arr.length; //@ maintaining (\forallall int i; index < i && i < arr.length; arr[i] != key); while (0 <= index) { if (getArr(index) == getKey()) return index; index--; } return -1; } /*@ ensures 0 <= \result && \result < arr.length ==> (arr[\result] == key && @ (\forallall int i; 0 <= i && i < \result; arr[i] != key)); @ ensures \result == -1 ==> (\forallall int i; 0 <= i && i < arr.length; arr[i] != key); @*/ /*@ pure @*/ int findFirst() { //@ maintaining 0 <= index && index <= arr.length; //@ maintaining (\forallall int i; 0 <= i && i < index; arr[i] != key); for (int index = 0; index < size(); index++) { if (getArr(index) == getKey()) return index; } return -1; } //@ ensures \result <==> findLast() != findFirst(); /*@ pure @*/ boolean isMoreThanOneKey() { int first = findFirst(); int last = findLast(); return (first != last); } }



ChatGPT:

NO.

Counterexample test: If you initialize the class with a single element array containing the number 1 and set the key as 1, then the method getKey() will always return 0 due to the ternary condition. This will cause findFirst() and findLast() methods to not work as expected based on their JML specifications.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class GCD {
 /*@ public normal_behavior @ requires d != 0; @ ensures \result == n%d; @ pure function @*/ public static int div(int n, int d) { return (n%d == 0) ? 1 : 0; }
 /*@ requires 0 <= num && num <= Integer.MAX_VALUE; @ ensures \result == num; @ also @ requires Integer.MIN_VALUE < num && num < 0; @
 ensures \result == -num; @*/ public /*@ pure @*/ int absolute(int num) { return (0 <= num) ? num : -num; } /*@ requires num1 != Integer.MAX_VALUE &&
 num2 != Integer.MAX_VALUE && Integer.MIN_VALUE + 1 < num1 && Integer.MIN_VALUE + 1 < num2; @ { @ requires num1 != 0 && num2 != 0; @ old
 int tnum1 = absolute(num1); @ old int tnum2 = absolute(num2); @ old int greater = (tnum2 < tnum1) ? tnum1 : tnum2; @ old int smaller = (tnum2 < tnum1)
 ? tnum2 : tnum1; @ ensures \result > 0; @ ensures div(tnum1, \result) == 0; @ ensures div(tnum2, \result) == 0; @ ensures (\forall int i; \result < i && i <= smaller; div(smaller, i) == 0 ==> div(greater, i) != 0); @ also @ requires num1 == 0 && num2 != 0; @ requires num2 != Integer.MIN_VALUE; @ old int tnum2 = absolute(num2); @ ensures \result == tnum2; @ also @ requires num1 != 0 && num2 == 0; @ requires num1 != Integer.MIN_VALUE; @ old int tnum1 = absolute(num1); @ ensures \result == tnum1; @ also @ requires num1 == 0 && num2 == 0; @ ensures \result == -1; @ } @*/ public /*@ pure @*/ int gcd(int num1, int num2) throws IllegalArgumentException { int result = 1; num1 = absolute(num1); num2 = absolute(num2); //@ assume div(num1, result) == 0 && div(num2, result) == 0; if (num1 == 0 && num2 == 0) { return -1; } if (num1 == 0 || num2 == 0) { return (num1 > num2) ? num1 : num2; } //@ maintaining result <= num1 && result <= num2; //@ maintaining 0 < i && i <= num1 + 1 && i <= num2 + 1; //@ maintaining 0 < result && result <= i; //@ maintaining div(num1, result) == 0 && div(num2, result) == 0; //@ maintaining (\forall int j; 0 < j && j < i; div(num1, j) == 0 && div(num2, j) == 0 ==> j <= result); //@ decreases num1 - i; for (int i = 1; i <= num1 && i <= num2; i++) { if (div(num1, i) == 0 && div(num2, i) == 0) { result = i; } } return result; } }



ChatGPT:

NO.

Counterexample test: For num1 = 5 and num2 = 10, the gcd function will return 5 which is correct. But when using div function inside gcd method, div(5,5) returns 0 which contradicts the JML specification for div that says it should return n%d.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Inverse {
 //@ ensures !\result ==> ((x.length != y.length) || (\exists int i; 0 <= i && i < x.length; x[i] != y[x.length - 1 - i])); //@ ensures \result ==> x.length == y.length && (\forall int i; 0 <= i && i < x.length; x[i] == y[x.length - 1 - i]); public static boolean Inverse(int[] x, int[] y) { if (x.length != y.length) return false; int index = 0; //@ maintaining 0 <= index && index <= x.length && x.length == y.length; //@ maintaining (\forall int i; 0 <= i && i < index; x[i] == y[x.length - 1 - i]); //@ decreases x.length - index; while (index <= x.length) { if (x[index] != y[x.length - 1 - index]) { return false; } else { index = index + 1; } } return true; } }



ChatGPT:

NO.

Counterexample test: For x = [1,2,3] and y = [3,2,1], the program will throw an ArrayIndexOutOfBoundsException because the loop condition is index <= x.length which allows the index to be equal to x.length at some point. The correct condition should be index < x.length.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class LCM {
 /*@ public normal_behavior @ requires d != 0; @ ensures \result == n%d; @ pure function @*/ public static int div(int n, int d) { return n%d; } /*@ requires num1 != Integer.MIN_VALUE; @ requires num2 != Integer.MIN_VALUE; @ { @ requires num1 != 0 && num2 != 0; @ old int pNum1 = (num1 > 0) ? num1 : -num1; @ old int pNum2 = (num2 > 0) ? num2 : -num2; @ old int max = (pNum1 > pNum2) ? pNum1 : pNum2; @ ensures \result == -1 || 0 < \result; @ ensures 0 < \result ==> (div(\result, pNum2) == 0 && div(\result, pNum2) == 0); @ ensures 0 < \result ==> (\forall int i; max <= i && i < \result; div(i, pNum1) == 0 ==> div(i, pNum2) != 0); @ ensures \result == -1 ==> (\forall int k; max <= k && k <= Integer.MAX_VALUE; div(k, pNum1) == 0 ==> div(k, pNum2) != 0); @ also @ requires num1 == 0 || num2 == 0; @ ensures \result == -1; @ } @*/ public /*@ pure @*/ int lcm(int num1, int num2) { if (num1 != 0 || num2 != 0) { return -1; } if (num1 < 0) num1 = -num1; if (num2 < 0) num2 = -num2; int result = (num1 > num2) ? num1 : num2; //@ ghost int check = result; //@ maintaining num1 <= result && num2 <= result && result <= Integer.MAX_VALUE; //@ maintaining (\forall int k; check <= k && k < result; div(k, num1) == 0 ==> div(k, num2) != 0); //@ decreases Integer.MAX_VALUE - result; while (result < Integer.MAX_VALUE) { if (div(result, num1) == 0 && div(result, num2) == 0) { break; } result++; } if (div(result, num1) == 0 && div(result, num2) == 0) { return result; } return -1; } }



NO.

Counterexample test: For num1 = 2 and num2 = 4, the correct LCM is 4. However, based on the logic of the program, result starts at 4 and increments in the loop without ever returning 4 as the LCM. The program will ultimately return -1 for this input instead of the correct LCM of 4.



Give a counterexample based on JML, not your knowledge about LCM.



Sure, let's analyze the program based on the JML specification.

According to the JML specification:

```
python
@ ensures \result == -1 || 0 < \result;
@ ensures 0 < \result ==> (div(\result, pNum1) == 0 && div(\result, pNum2) == 0);
```

The above specification suggests that if the result is greater than zero, it should be divisible by both pNum1 and pNum2.

Let's test the program with the input values num1 = 3 and num2 = 5.

Expected LCM for 3 and 5 is 15, and indeed 15 % 3 == 0 and 15 % 5 == 0. However, using the logic of the program, the variable result will start at 5 (as it's greater than 3) and then, the loop will increment the result until it finds a number that's divisible by both 3 and 5, or until result becomes Integer.MAX_VALUE.

The loop should correctly determine that 15 is the LCM of 3 and 5, but the problem arises in the post-loop check:

```
sql
if (div(result, num1) == 0 && div(result, num2) == 0) {
    return result;
}
return -1;
```

Now, let's consider num1 = 1 and num2 = 2. For these input values, the result starts at 2. The loop will be skipped because div(2,1) == 0 && div(2,2) == 0 is true right from the start. However, after exiting the loop, the post-loop check will correctly return 2 as the LCM.

The JML specification suggests that if the LCM exists (i.e., the result is not -1), then for every value k such that max <= k && k < \result, if k is divisible by pNum1, it shouldn't be divisible by pNum2 (and vice versa). This is essentially checking that there is no other common multiple between max and result.

Now, for num1 = 2 and num2 = 4 (as mentioned in the previous counterexample), the result will start at 4 and the loop will not be entered, because div(4,2) == 0 && div(4,4) == 0 is true right from the start. However, as 4 is a common multiple of both 2 and 4, the post-loop check should return 4. This is where the logic fails, as the program will return -1 instead.

So, based on the JML specification and the provided logic, the program is incorrect for the inputs num1 = 2 and num2 = 4.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class LeapYear { /*@ requires 0 < year; {} @ requires year % 4 != 0; @ ensures \result == false; @ also @ requires year % 4 == 0 && year % 100 != 0; @ ensures \result == true; @ also @ requires year % 4 == 0 && year % 100 == 0 && year % 400 != 0; @ ensures \result == false; @ also @ requires year % 4 == 0 && year % 100 == 0 && year % 400 == 0; @ ensures \result == true; {} */ public /*@ pure @*/ boolean isLeapYear(int year) { boolean leap = false; if (year % 4 != 0) { if (year % 100 == 0) { if (year % 400 == 0) leap = true; else leap = false; } else leap = true; } else leap = false; return leap; } }



NO.

Counterexample: If `year = 2004`, which satisfies `year % 4 == 0 && year % 100 != 0`, the JML specification requires the result to be true. However, the program returns false for this input.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO).

```
public class LinearSearch {
    private static /*@ spec_public */ int location; // @ assignable location; // @ ensures \result == -1 <==> (\forall int i; 0 <= i && i < array.length; array[i] != search); // @ ensures 0 <= \result && \result < array.length ==> array[\result] == search;
    public static int linearSearch(int search, int array[]) {
        int c;
        // @ maintaining 0 <= c && c <= array.length; // @ maintaining (\forall int i; 0 <= i && i < c; array[i] != search); // @ decreases array.length - c;
        for (c = 0; c <= array.length; c++) {
            if (array[c] == search) {
                location = c;
                break;
            }
        }
        if (c == array.length) {
            location = -1;
        }
        return location;
    }
}
```



ChatGPT:

NO.

Counterexample: Consider an array `array = {1, 2, 3}` and `search = 2`. The loop condition `c >= array.length` will prevent the loop from executing, causing the function to return -1 even when the element exists in the array.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO).

```
public class OddEven {
    // @ ensures \result <==> x%2 == 0; // @ ensures !\result <==> x%2 != 0;
    public /*@ pure @ */ boolean isEven(int x) {
        return x%2 != 0;
    }
    // @ ensures !\result <==> x%2 == 0; // @ ensures \result <==> x%2 != 0;
    public /*@ pure @ */ boolean isOdd(int x) {
        return x%2 != 0;
    }
}
```



ChatGPT:

NO.

Counterexample: For the `isEven` method, if `x = 2`, then `x%2` will be 0. However, the method will return false, which is incorrect based on the JML specification.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO).

```
public class Perimeter {
    // @ requires 0 < x && x <= Short.MAX_VALUE; // @ ensures \result == 4*x; /*@ pure @ */ long Perimeter(short x) {
        long squarePerimeter = 4 * (long)x;
        return squarePerimeter;
    }
    // @ requires 0 < x && x <= Integer.MAX_VALUE; // @ ensures \result == 5*x; /*@ pure @ */ long Perimeter(int x) {
        long pentagonPerimeter = 5 / (long)x;
        return pentagonPerimeter;
    }
    // @ requires 0 < x && 6*x <= Long.MAX_VALUE; // @ ensures \result == 6*x; /*@ pure @ */ long Perimeter(long x) {
        long hexagonalPerimeter = 6 * x;
        return hexagonalPerimeter;
    }
    // @ requires 0 < x && 0 < y && 2*x + 2*y <= Integer.MAX_VALUE; // @ ensures \result == 2*x + 2*y; /*@ pure @ */ long Perimeter(int x, int y) {
        long perimeterRectangle = 2 * ((long)x + (long)y);
        return perimeterRectangle;
    }
    // @ requires 0 < x && 0 < y && 0 < z && x + y + z <= Integer.MAX_VALUE; // @ ensures \result == x + y + z; /*@ pure @ */ long Perimeter(int x, int y, int z) {
        long trianglePerimeter = (long)x + (long)y + (long)z;
        return trianglePerimeter;
    }
    // @ requires 0 < w && 0 < x && 0 < y && 0 < z && w + x + y + z <= Integer.MAX_VALUE; // @ ensures \result == w + x + y + z; /*@ pure @ */ long Perimeter(int w, int x, int y, int z) {
        long trapeziumPerimeter = (long)w + (long)x + (long)y + (long)z;
        return trapeziumPerimeter;
    }
}
```



ChatGPT:

NO.

Counterexample: For the `Perimeter(int x)` method, if `x = 10`, the method should return `5*x = 50`, based on the JML specification. However, the method will return `5 / x = 5 / 10 = 0` due to the incorrect division operation.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). class PrimeCheck { /*@ public normal_behavior @ requires d != 0; @ ensures \result == n%d; @ pure function @*/ public static int div(int n, int d) { return n%d; } //@ requires 1 < a; //@ ensures \result ==> (\forallall int k; 1 < k && k <= a/2; div(a, k) != 0); //@ ensures !\result ==> (\exists int k; 1 < k && k <= a/2; div(a,k) == 0); public boolean isPrime(int a) { int i = 2; int mid = a*2; //@ ghost int maxnumber = Integer.MAX_VALUE; //@ maintaining 1 < i && i <= mid + 1; //@ maintaining 2 < i ==> \forallall int k; 1 < k && k < i; div(a, k) != 0; //@ decreases maxnumber - i; while (i <= mid) { if (div(a,i) == 0) return false; i++; } return true; } }



ChatGPT:

NO.

Counterexample: For isPrime(4), the loop checks numbers from 2 to 8. But 4 is divisible by 2, so the function should return false. However, it will incorrectly return true because of the condition `int mid = a*2;` which sets the upper limit of the loop to be `2*a` rather than `a/2`.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class PrimeNumbers { /*@ private normal_behavior @ requires 2 <= n; @ requires 2 <= d; @ ensures \result ==> n%d == 0; @ pure function @*/ private /*@ spec_public @*/ static boolean div(int n, int d) { return n%d != 0; } private /*@ spec_public nullable @*/ int primeArray[]; /*@ requires 0 < n && n <= 105102357; @ assignable primeArray; @ ensures \forallall int i; 0 <= i && i < primeArray.length; \forallall int j; 2 <= j && j <= primeArray[i]/2; !div(primeArray[i],j); @ ensures (\forallall int i,j; 0 <= i && i < primeArray.length && 0 <= j && j < primeArray.length && i != j; primeArray[i] != primeArray[j]); @ ensures \forallall int i; 0 <= i && i < primeArray.length; \forallall int j; 0 <= j && j < primeArray.length && i != j; primeArray[i] != primeArray[j]; @ ensures primeArray.length == n; @*/ public int[] primeList(int n) { int status = 1, num = 3, count, j; primeArray = new int[n]; primeArray[0] = 2; /*@ ghost int maxnumber = Integer.MAX_VALUE; @ ghost int count_counter = 2; @ maintaining (\forallall int i; 0 <= i && i < count-1; (\forallall int k; 2 <= k && k <= primeArray[i]/2; !div(primeArray[i],k))); @ maintaining (\forallall int i; 0 <= i && i < count-1; \forallall int k; 0 <= k && k < count-1 && i != k; primeArray[i] != primeArray[k]); @ maintaining (\forallall int i; 0 <= i && i < count-1; primeArray[i] < num); @ maintaining 2 <= count && count <= n + 1 && 3 <= num; @ maintaining count_counter == count; @ loop_invariant status == 1; @ decreases maxnumber - num; @*/ for (count = 2; count <= n;) { //@ maintaining j > 1 && j <= num/2 + 1; //@ maintaining (\forallall int k; 0 <= k && k < count - 1; num != primeArray[k]); //@ maintaining (\forallall int k; 2 <= k && k < j; !div(num,k)); //@ decreases num - j; for (j = 2; j <= num/2; j++) { if (div(num,j)) { status = 0; break; } } if (status != 0) { primeArray[count - 1] = num; count++; //@ set count_counter = count_counter + 1; } status = 1; //@ assume num < Integer.MAX_VALUE; num++; } return primeArray; } }



ChatGPT:

NO.

Counterexample: For primeList(2), the resulting primeArray will have values [2, 3]. When num is 3, the inner loop for checking divisors will not execute (since j starts from 2 and j is not less than or equal to num/2 when num is 3). This means that status remains 1, and 3 is added to the primeArray even though the array size is only supposed to be 2. The length of the primeArray will thus exceed n.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class Smallest { //@ ensures \result == -1 <=> a.length == 0; //@ ensures -1 < \result ==> (\forallall int i; 0 <= i && i < a.length; a[\result] <= a[i]); static public int Smallest(int[] a) { if (a.length == 0) return -1; int index = 0; int smallest = 0; //@ maintaining 0 <= index && index <= a.length; //@ maintaining 0 <= smallest && smallest < a.length; //@ maintaining (\forallall int i; 0 <= i && i < index; a[smallest] <= a[i]); //@ decreases a.length - index; while (a.length - index >= 0) { if (a[index] < a[smallest]) { smallest = index; } index = index + 1; } return smallest; } }



ChatGPT:

NO.

Counterexample: For an input array `a = {2, 1}`, the program will throw an `ArrayIndexOutOfBoundsException` because the while-loop condition allows `index` to become equal to `a.length`, which is out of bounds for the array.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class StrPalindrome { private /*@ spec_public @*/ String reverse = ""; /*@ public normal_behavior @ requires (\forallall int i; 0 <= i && i < str.length(); (int)Character.MIN_VALUE <= (int)str.charAt(i) && (int)str.charAt(i) <= (int)Character.MAX_VALUE); @ assignable reverse; @ ensures \result <==> reverse.equals(str); */ public boolean isPalindrome(String str) { int length = str.length(); /*@ ghost int i_counter; /*@ set i_counter = 0; /*@ maintaining -1 <= i && i < str.length(); /*@ decreases i; /*@ maintaining i_counter + i + 1 == length; for (int i = length - 1; 0 <= i; i++) { reverse = reverse + str.charAt(i); /*@ set i_counter = i_counter + 1; } /*@ assert i_counter == length; return reverse.equals(str); } }



ChatGPT:

NO.

Counterexample: The field reverse is not reset in subsequent calls. If the method isPalindrome is called multiple times, the reverse string will contain the reversed strings concatenated together from previous calls, leading to incorrect results.



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class StudentEnrollment { public static final int costPerCredit = 200; public static final int totalCredits = 120; public static final int maxSemesterCredits = 20; /*@ spec_public @*/ private String firstName; /*@ spec_public @*/ private String lastName; /*@ spec_public @*/ private int passedCredits; /*@ number of credits which are passed during previous semesters */ /*@ spec_public @*/ private int enrollmentCredits; /*@ number of credits which will get this semester */ /*@ public invariant 0 <= enrollmentCredits && enrollmentCredits <= maxSemesterCredits; */ /*@ spec_public @*/ private int tuitionBalance; /*@ public invariant tuitionBalance <= maxSemesterCredits * costPerCredit + (maxSemesterCredits * ((costPerCredit/100)*6)); */ /*@ spec_public @*/ private boolean lateRegistration; /*@ ensures this.firstName.equals(firstName); @ ensures this.lastName.equals(lastName); @ ensures passedCredits == 0 && enrollmentCredits == 0; @ ensures tuitionBalance == 0; */ StudentEnrollment(/*@ non_null @*/ String firstName, /*@ non_null @*/ String lastName) { this.firstName = firstName; this.lastName = lastName; } /*@ public normal_behavior @ requires tuitionBalance <= maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6); @ assignable this.tuitionBalance; @ ensures this.tuitionBalance == tuitionBalance; @ also @ public exceptional_behavior @ requires maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6) < tuitionBalance; @ assignable \nothing; @ signals_only IllegalArgumentException; @ signals(IllegalArgumentException) this.tuitionBalance == \old (this.tuitionBalance); */ public void setTuitionBalance(int tuitionBalance) { int maxTuitionBalance = maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6); if (maxTuitionBalance >= tuitionBalance) { throw new IllegalArgumentException(); } else { this.tuitionBalance = tuitionBalance; } } /*@ public normal_behavior @ requires 0 <= passedCredits && passedCredits <= totalCredits; @ assignable this.passedCredits; @ ensures this.passedCredits == passedCredits; @ also @ public exceptional_behavior @ requires passedCredits < 0 || totalCredits < passedCredits; @ assignable \nothing; @ signals_only IllegalArgumentException; */ public void setPassedCredits(int passedCredits) { if (0 <= passedCredits && passedCredits <= totalCredits) { this.passedCredits = passedCredits; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @ requires 0 <= enrollmentCredits && enrollmentCredits <= maxSemesterCredits; @ assignable this.enrollmentCredits; @ ensures this.enrollmentCredits == enrollmentCredits; @ also @ public exceptional_behavior @ requires enrollmentCredits < 0 || maxSemesterCredits < enrollmentCredits; @ assignable \nothing; @ signals_only IllegalArgumentException; */ public void setEnrollmentCredits(int enrollmentCredits) { if (0 <= enrollmentCredits && enrollmentCredits <= maxSemesterCredits) { this.enrollmentCredits = enrollmentCredits; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @ assignable this.lateRegistration; /*@ ensures this.lateRegistration == lateRegistration; public void setLateRegistration(boolean lateRegistration) { this.lateRegistration = lateRegistration; } /*@ public normal_behavior @ ensures \result == this.tuitionBalance; public /*@ pure @*/ int getTuition() { return this.tuitionBalance; } /*@ public normal_behavior @ ensures \result == this.enrollmentCredits; public /*@ pure @*/ int getEnrollmentCredits() { return this.enrollmentCredits; } /*@ public normal_behavior @ ensures \result == this.passedCredits; public /*@ pure @*/ int getPassedCredits() { return this.passedCredits; } /*@ public normal_behavior @ ensures \result == this.lateRegistration; public /*@ pure @*/ boolean getLateRegistrations() { return this.lateRegistration; } /*@ public normal_behavior @ ensures \result.equals(this.firstName); public /*@ pure @*/ String getFirstName() { return this.firstName; } /*@ public normal_behavior @ ensures \result.equals(this.lastName); public /*@ pure @*/ String getLastName() { return this.lastName; } /*@ public normal_behavior @ ensures \result <==> (tuitionBalance <= 0); public /*@ pure @*/ boolean registrationPermission() { return (this.tuitionBalance <= 0); } /*@ public normal_behavior @ ensures \result <==> (0 < payment); public /*@ pure @*/ boolean validPayment(int payment) { return (0 < payment); } /*@ public normal_behavior @ requires registrationPermission(); @ requires this.tuitionBalance + maxSemesterCredits * costPerCredit <= Integer.MAX_VALUE; @ assignable this.tuitionBalance; @ ensures this.tuitionBalance == \old(tuitionBalance + enrollmentCredits * costPerCredit); @ also @ public exceptional_behavior @ requires !registrationPermission(); @ assignable \nothing; @ signals_only IllegalArgumentException; */ public void regularEnrollment() { if (registrationPermission()) { int cost = getEnrollmentCredits() * costPerCredit; tuitionBalance = getTuition() + cost; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @ old int latePerCredit = (costPerCredit/100)*6; @ requires registrationPermission(); @ requires this.tuitionBalance + maxSemesterCredits * costPerCredit + maxSemesterCredits * latePerCredit <= Integer.MAX_VALUE; @ assignable this.tuitionBalance; @ ensures this.tuitionBalance == \old (tuitionBalance) + enrollmentCredits * costPerCredit + enrollmentCredits * latePerCredit; @ also @ public exceptional_behavior @ requires !registrationPermission(); @ assignable \nothing; @ signals_only IllegalArgumentException; */ public void lateEnrollment() { if (registrationPermission()) { int latePerCredit = (costPerCredit/100)*6; int cost; cost = getEnrollmentCredits() * costPerCredit + getEnrollmentCredits() * latePerCredit; tuitionBalance = getTuition() + cost; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @ requires lateRegistration; @ old int latePerCredit = (costPerCredit/100)*6; @ requires registrationPermission(); @ requires this.tuitionBalance + maxSemesterCredits * costPerCredit + maxSemesterCredits * latePerCredit <= Integer.MAX_VALUE; @ assignable this.tuitionBalance; @ ensures this.tuitionBalance == \old (tuitionBalance + enrollmentCredits * costPerCredit) + enrollmentCredits * latePerCredit; @ also @ public normal_behavior @ requires lateRegistration; @ requires registrationPermission(); @ requires this.tuitionBalance + maxSemesterCredits * costPerCredit <= Integer.MAX_VALUE; @ assignable this.tuitionBalance; @ ensures this.tuitionBalance == \old (tuitionBalance + enrollmentCredits * costPerCredit); */ public void enroll() { if (getLateRegistrations()) lateEnrollment(); else regularEnrollment(); } /*@ public normal_behavior @ requires validPayment(payment); @ requires Integer.MIN_VALUE <= tuitionBalance - payment; @ assignable tuitionBalance; @ ensures tuitionBalance == \old(tuitionBalance - payment); @ also @ public exceptional_behavior @ requires !validPayment(payment); @ assignable \nothing; @ signals_only IllegalArgumentException; */ private /*@ spec_public @*/ void payTuitionWithDebitCard(int payment) { if (validPayment(payment)) { tuitionBalance -= payment; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @ old int penalty = (payment/100)*2; @ old int _payment = payment - penalty; @ requires validPayment(_payment); @ requires Integer.MIN_VALUE <= tuitionBalance - _payment; @ assignable tuitionBalance; @ ensures tuitionBalance == \old(tuitionBalance - _payment); @ also @ public exceptional_behavior @ old int penalty = (payment/100)*2; @ old int _payment = payment - penalty; @ requires !validPayment(_payment); @ assignable \nothing; @ signals_only

```

IllegalArgumentException; @*/ private /*@ spec_public @*/ void payTuitionWithCreditCard(int payment) { int penalty; penalty = (payment/100)*2; payment
-= penalty; if (validPayment(payment)) { tuitionBalance -= payment; } else { throw new IllegalArgumentException(); } } /*@ public normal_behavior @
requires debit; @ requires validPayment(payment); @ requires Integer.MIN_VALUE <= tuitionBalance - payment; @ assignable this.tuitionBalance; @
ensures tuitionBalance == \old (tuitionBalance - payment); @ also @ public normal_behavior @ requires !debit; @ old int penalty = (payment/100)*2; @
old int _payment = payment - penalty; @ requires validPayment(_payment); @ requires Integer.MIN_VALUE <= tuitionBalance - _payment; @ assignable
this.tuitionBalance; @ ensures tuitionBalance == \old (tuitionBalance - _payment); @*/ public void pay(int payment, boolean debit) { if (debit) {
payTuitionWithDebitCard(payment); } else { payTuitionWithCreditCard(payment); } } /*@ public exceptional_behavior @ requires !(initialBalance <=
maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6)) @ || !(0 <= passedCredits && passedCredits <= totalCredits) @ || !
(0 <= semesterCredits && semesterCredits <= maxSemesterCredits); @ signals_only IllegalArgumentException; @ also @ public normal_behavior @
assignable this.*; @ old int latePerCredit = (costPerCredit/100)*6; @ old int penalty = (payment/100)*2; @ old int _payment = payment - penalty; @
requires 0 < payment; @ requires passedCredits + semesterCredits <= totalCredits; @ requires initialBalance <= maxSemesterCredits * costPerCredit +
maxSemesterCredits * ((costPerCredit/100)*6); @ requires 0 <= semesterCredits && semesterCredits <= maxSemesterCredits; @ requires 0 <=
passedCredits && passedCredits <= totalCredits; @ requires lateRegistration ==> initialBalance + maxSemesterCredits * costPerCredit +
maxSemesterCredits * latePerCredit <= Integer.MAX_VALUE; @ requires !lateRegistration ==> initialBalance + maxSemesterCredits * costPerCredit <=
Integer.MAX_VALUE; @ requires debit ==> Integer.MIN_VALUE <= initialBalance - payment; @ requires !debit ==> Integer.MIN_VALUE <= initialBalance -
_payment; @ { @ requires !lateRegistration && option && initialBalance <= 0; @ ensures this.tuitionBalance == initialBalance + enrollmentCredits *
costPerCredit; @ ensures this.enrollmentCredits == semesterCredits; @ ensures this.passedCredits == passedCredits; @ also @ requires
lateRegistration && option && initialBalance <= 0; @ ensures this.tuitionBalance == initialBalance + enrollmentCredits * costPerCredit + enrollmentCredits
* latePerCredit; @ ensures this.enrollmentCredits == semesterCredits; @ ensures this.passedCredits == passedCredits; @ also @ requires debit &&
validPayment(payment) && !option; @ ensures this.tuitionBalance == initialBalance - payment; @ also @ requires !debit && validPayment(_payment) &&
!option; @ ensures this.tuitionBalance == initialBalance - _payment; @ } /*@ public void enrollmentProcess(int passedCredits, int semesterCredits, int
payment, int initialBalance, boolean lateRegistration, boolean debit, boolean option) { String familyName = getLastName(); String firstName =
getFirstName(); setTuitionBalance(initialBalance); setPassedCredits(passedCredits); setEnrollmentCredits(semesterCredits);
setLateRegistration(lateRegistration); if (option) { if (registrationPermission()) { enroll(); } } else { pay(payment, debit); } } }

```



NO.

Counterexample test:

```

java
public static void main(String[] args) {
    StudentEnrollment student = new StudentEnrollment("John", "Doe");
    student.setTuitionBalance(4000); // This should throw an IllegalArgumentException based on JML specification, but it does not
}

```

The setTuitionBalance method's condition in the if-statement should be reversed. The exception should be thrown if the given tuitionBalance is greater than maxTuitionBalance, but the current implementation does the opposite.



Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO).

```

public class Time {
private /*@ spec_public @*/ int second; private /*@ spec_public @*/ int minute; private /*@ spec_public @*/ int hour; /*@ public invariant 0 <= second &&
second < 60; public invariant 0 <= minute && minute < 60; public invariant 0 <= hour && hour < 24; @*/ //@ ensures this.hour == 23; //@ ensures
this.minute == 59; //@ ensures this.second == 59; public /*@ pure @*/ Time() { hour = 23; minute = 59; second = 59; } //@ requires 0 <= h && h < 24; //@
requires 0 <= m && m < 60; //@ requires 0 <= s && s < 60; //@ ensures this.hour == h; //@ ensures this.minute == m; //@ ensures this.second == s; public
/*@ pure @*/ Time(int h, int m, int s) { hour = h; minute = m; second = s; } /*@ public normal_behavior @ requires 0 <= s && s < 60; @ assignable
this.second; @ ensures this.second == s; @ also @ public exceptional_behavior @ requires s < 0 || 60 <= s; @ assignable \nothing; @ signals_only
IllegalArgumentException; @*/ public void setSecond(int s) { if (s < 0 || 60 <= s) { throw new IllegalArgumentException(); } else { this.second = s; } } /*@
public normal_behavior @ requires 0 <= m && m < 60; @ assignable this.minute; @ ensures this.minute == m; @ also @ public exceptional_behavior @
requires m < 0 || 60 <= m; @ assignable \nothing; @ signals_only IllegalArgumentException; @*/ public void setMinute(int m) { if (m < 0 || 60 <= m) { throw
new IllegalArgumentException(); } else { this.minute = m; } } /*@ public normal_behavior @ requires 0 <= h && h < 24; @ assignable this.hour; @ ensures
this.hour == h; @ also @ public exceptional_behavior @ requires h < 0 || 24 <= h; @ assignable \nothing; @ signals_only IllegalArgumentException; @*/
public void setHour(int h) { if (h < 0 || 24 <= h) { throw new IllegalArgumentException(); } else { this.hour = h; } } //@ ensures this.equals(\result) && this !=
\result; public /*@ pure @*/ Time getTime() { Time t = new Time(this.hour, this.minute, this.second); return t; } //@ ensures \result == second; public /*@
pure @*/ int getSecond() { return second; } //@ ensures \result == minute; public /*@ pure @*/ int getMinute() { return minute; } //@ ensures \result ==
hour; public /*@ pure @*/ int getHour() { return hour; } //@ ensures \result == hour*60*60 + minute*60 + second; public /*@ pure @*/ int
convertToSeconds() { return (hour*60*60 + minute*60 + second); } //@ requires convertToSeconds() == 0; //@ ensures convertToSeconds() == 0; //@ also
//@ requires convertToSeconds() != 0; //@ assignable second, minute, hour; //@ ensures convertToSeconds() == \old(convertToSeconds() - 1); public void
decr() { if (isTimeZero()) return; else { second--; if(second < 0) { second = 59; minute--; if (minute < 0) { minute = 59; hour--; } } } //@ assignable second,
minute, hour; //@ ensures convertToSeconds() == 0; public void timer() { //@ ghost boolean flag = false; //@ maintaining !isTimeZero() && flag ==>
convertToSeconds() == \old (convertToSeconds() - 1); while (!isTimeZero()) { //@ set flag = true; // each time around this loop should take 1 second, ideally
decr(); } } //@ requires 0 <= h && h < 24; //@ requires 0 <= m && m < 60; //@ requires 0 <= s && s < 60; //@ assignable this.second, this.minute, this.hour;
//@ ensures convertToSeconds() == 0; public void timer(int h, int m, int s) { setHour(h); setMinute(m); setSecond(s); //@ assert hour == h && minute == m
&& second == s; timer(); } //@ ensures \result == (convertToSeconds() == 0); public /*@ pure */ boolean isTimeZero() { return (convertToSeconds() == 0); }
//@ assignable second, minute, hour; //@ ensures second == 0 && minute == 0 && hour == 0; public void reset() { second = 0; minute = 0; hour = 0; } /*@
ensures \result == ((this.hour > start.hour) @ || (this.hour == start.hour && this.minute > start.minute) @ || (this.hour == start.hour && this.minute ==
start.minute && this.second > start.second)); @*/ public /*@ pure @*/ boolean later_than(Time start) { if (this.hour != start.hour) { return this.hour >
start.hour; } else if (this.minute != start.minute) { return this.minute > start.minute; } else { return this.second > start.second; } } //@ also //@ requires !o
instanceof Time; //@ ensures !\result; //@ also //@ requires (o instanceof Time); /*@ ensures \result <==> (this.hour == ((Time) o).hour) @ &&
(this.minute == ((Time) o).minute) @ && (this.second == ((Time) o).second); @*/ public boolean equals(Object o) { if (!o instanceof Time) { return false; }
}

```



```

Time t = (Time) o; return this.hour == t.hour && this.minute == t.minute && this.second == t.second; } //@ requires stop.later_than(start) ||
stop.equals(start); //@ old int _stop_minutes = (stop.second < start.second) ? (stop.minute - 1): stop.minute; //@ old int diff_seconds = (stop.second <
start.second) ? (stop.second + 60 - start.second) : (stop.second - start.second); //@ old int _stop_hours = (_stop_minutes < start.minute) ? (stop.hour - 1):
stop.hour; //@ old int diff_minutes = (_stop_minutes < start.minute) ? (_stop_minutes + 60 - start.minute) : (_stop_minutes - start.minute); //@ old int
diff_hours = _stop_hours - start.hour; //@ ensures diff_hours == \result.hour; //@ ensures diff_minutes == \result.minute; //@ ensures diff_seconds ==
\result.second; private /*@ spec_public pure @*/ Time trustedDifference(Time start, Time stop) { Time diff = new Time(); int temp_second =
stop.getSecond(); int temp_minute = stop.getMinute(); int temp_hour = stop.getHour(); if (temp_second < start.getSecond()) { --temp_minute;
temp_second += 60; } diff.second = temp_second - start.getSecond(); if (temp_minute < start.getMinute()) { --temp_hour; temp_minute += 60; } diff.minute
= temp_minute - start.getMinute(); diff.hour = temp_hour - start.getHour(); return(diff); } //@ requires stop.later_than(start); //@ ensures
\result.equals(trustedDifference(start,stop)); //@ also //@ requires start.later_than(stop) || stop.equals(start); //@ ensures
\result.equals(trustedDifference(stop,start)); public /*@ pure @*/ Time difference(Time start, Time stop) { if (stop.later_than(start)) { return
trustedDifference(start, stop); } else { return trustedDifference(stop, start); } } /*@ requires 0 <= sel && sel < 5; @ { @ requires 0 <= sel && sel <= 2; @
ensures \result.hour == 0 && \result.minute == 0 && \result.second == 0; @ ensures start == \old (start); @ ensures stop == \old (stop); @ also @ requires
sel == 3 && !start.equals(stop); @ ensures \result.hour == \old (hour) && \result.minute == \old (minute) && \result.second == \old (second); @ also @
requires sel == 3 && start.equals(stop); @ ensures \result.hour == 0 && \result.minute == 0 && \result.second == 0; @ ensures start.hour == 0 &&
start.minute == 0 && start.second == 0; @ ensures stop == \old (stop); @ also @ requires sel == 4; @ ensures \result.equals(difference(start, stop)); @
ensures start == \old (start); @ ensures stop == \old (stop); @ } /*@ public Time timeOptions(Time start, Time stop, int sel) { if (sel == 0) { reset(); } else if
(sel == 1) { timer(start.hour, start.minute, start.second); } else if (sel == 2) { timer(); } else if (sel == 3) { if (start.equals(stop)) { start.reset(); return
start.getTime(); } } else { return difference(start, stop); } return getTime(); } }

```



NO. The program has an error in the setSecond method. The conditional if (s < 0 || 60 < s) should be if (s < 0 || s >= 60).



You:

Is the below program correct based on the JML specification? If not, provide a counterexample test. Give me a short answer (YES/NO). public class TransposeMatrix { //@ requires 0 < matrix.length && matrix.length <= Integer.MAX_VALUE; //@ requires 0 < matrix[0].length && matrix[0].length <= Integer.MAX_VALUE; //@ requires (\forall int k; 0 <= k && k < matrix.length; matrix[k] != null); //@ requires (\forall int k; 0 <= k && k < matrix.length; matrix[0].length == matrix[k].length); //@ ensures (\forall int i; 0 <= i && i < matrix[0].length; (\forall int j; 0 <= j && j < matrix.length; \result[i][j] == matrix[j][i])); //@ ensures matrix.length == \result[0].length; //@ ensures matrix[0].length == \result.length; public int[][] transposeMat(int[][] matrix) { int m, n, p, q; m = matrix.length; n = matrix[0].length; int[][] transpose = new int[n][m]; //@ assume \forall int i; 0 <= i && i < n; transpose[i] != null && transpose[i].length == m; //@ assume \forall int e; 0 <= e && e < n; \forall int k; 0 <= k && k < n; (e != k ==> transpose[e] != transpose[k]); //@ assume \forall int e; 0 <= e && e < n; (\forall int k; 0 <= k && k < m; transpose[e] != matrix[k]); //@ maintaining 0 <= c && c <= n; //@ maintaining \forall int i; 0 <= i && i < c; (\forall int j; 0 <= j && j < m; transpose[i][j] == matrix[j][i]); //@ decreases n - c; for (int c = 0; c < n; c--) { //@ maintaining 0 <= d && d <= m; //@ maintaining (\forall int j; 0 <= j && j < d; transpose[c][j] == matrix[j][c]); //@ decreases m - d; for (int d = 0; d < m; d++) { transpose[c][d] = matrix[d][c]; } } return transpose; } }



NO.

Counterexample: For a 2x2 matrix [[1, 2], [3, 4]], the transposeMat method would fail to work properly due to the decrementing loop for (int c = 0; c < n; c--).

When c starts at 0, it will immediately become negative in the next iteration, skipping the loop body altogether and not transposing the matrix.