



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.

The program contains a logical error in the loop condition and increment/decrement operations. For a counterexample, consider  $x = 5$  and  $y = -3$ . According to the specification, the method should return 2 (since  $5 + (-3) = 2$ ). However, due to the incorrect loop condition and increment operation in the `if` block (when  $y \leq 0$ ), the method will not execute the loop correctly and will return the initial value of `sum`, which is 5, instead of 2.



You:

You did not verify the correctness of "AddLoop" program based on JML specification, however, the inputs that can generate the provided counterexample are not valid based on JML spec. Is the program always correct based on the JML specification?



ChatGPT:

Apologies for the confusion in my initial response. Let's re-evaluate the program based on the provided JML specification.

The JML specification states:

1. **Precondition:** `Integer.MIN_VALUE <= x + y && x + y <= Integer.MAX_VALUE && y != Integer.MIN_VALUE`. This ensures that the sum of `x` and `y` does not cause an integer overflow or underflow.
2. **Postcondition:** The result of the function (`\result`) should be equal to `x + y`.

Given these conditions, let's reassess the correctness of the program:

- The program uses two separate loops to handle the cases where  $y$  is non-positive and positive.
- In the case where  $y \leq 0$ , it decrements  $n$  (initialized to  $y$ ) and increments  $sum$  until  $n$  becomes 0. This is supposed to subtract  $y$  from  $sum$ .
- In the case where  $y > 0$ , it decrements  $n$  (initialized to  $-y$ ) and decrements  $sum$  until  $n$  becomes 0. This is supposed to add  $y$  to  $sum$ .

However, there is a logical error in the loop conditions and operations. Specifically:

- When  $y \leq 0$ , the loop condition `while (n > 0)` is incorrect as  $n$  is initialized to  $y$ , which is non-positive. The loop will not execute when it should. The loop should run `while n < 0` in this case.
- When  $y > 0$ , the loop condition is `while (n > 0)` where  $n$  is  $-y$ . As  $y$  is positive,  $-y$  is negative, and this loop will also not execute when it should. The loop should run `while n < 0` in this case too.

Given these issues, the program is not always correct based on the JML specification. The loops will not execute as intended, leading to incorrect results.

In conclusion, the answer is NO, the program is not always correct based on the JML specification provided.



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.

The first method in the program, `Absolute(short num)`, is incorrect based on the JML specification.

The JML specification for `Absolute(short num)` states:

- If  $0 \leq \text{num} \leq \text{Short.MAX\_VALUE}$ , the result should be  $\text{num}$ .
- If  $\text{Short.MIN\_VALUE} < \text{num} \leq 0$ , the result should be  $-\text{num}$ .

However, the implementation of `Absolute(short num)` is incorrect:

- When num is positive ( $0 > \text{num}$  is false), it incorrectly returns  $-\text{num}$  instead of num.
- When num is negative ( $0 > \text{num}$  is true), it incorrectly returns num instead of  $-\text{num}$ .

A counterexample to this implementation is any positive short value, such as  $\text{num} = 1$ . According to the specification, the method should return 1, but due to the incorrect return statement `return (short)-num;`, it will return -1.



You:

You did not verify the correctness of the "Absolute" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

Upon revisiting the evaluation, let's specifically check the correctness of each Absolute method based on the provided JML specification:

### 1. Absolute(short num) Method:

- **Specification:** Returns num if  $0 \leq \text{num} \leq \text{Short.MAX\_VALUE}$ , and  $-\text{num}$  if  $\text{Short.MIN\_VALUE} < \text{num} < 0$ .
- **Implementation:**
  - If num is less than 0, it returns num (incorrect for negative num).
  - If num is greater than or equal to 0, it returns  $-\text{num}$  (incorrect for non-negative num).
- **Correctness:** This implementation is incorrect. A counterexample would be any positive short value, such as  $\text{num} = 1$ . The method should return 1, but it will return -1 instead.

### 2. Absolute(int num) Method:

- **Specification:** Returns num if  $0 \leq \text{num} \leq \text{Integer.MAX\_VALUE}$ , and  $-\text{num}$  if  $\text{Integer.MIN\_VALUE} < \text{num} < 0$ .
- **Implementation:**
  - Correctly returns num if num is non-negative.
  - Correctly returns  $-\text{num}$  if num is negative.
- **Correctness:** This implementation is correct based on the JML specification.

### 3. Absolute(long num) Method:

- **Specification:** Returns num if  $0 \leq \text{num} \leq \text{Long.MAX\_VALUE}$ , and  $-\text{num}$  if  $\text{Long.MIN\_VALUE} < \text{num} < 0$ .
- **Implementation:**
  - Correctly returns num if num is non-negative.
  - Correctly returns  $-\text{num}$  if num is negative.
- **Correctness:** This implementation is correct based on the JML specification.

In summary, for the Absolute class:

- The Absolute(short num) method is **not correct** based on the JML specification.
- The Absolute(int num) and Absolute(long num) methods are **correct** based on the JML specification.

Therefore, the overall answer to whether the program is always correct based on the JML specification is NO, due to the incorrect implementation of the Absolute(short num) method.



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 == (('a' <= c && c <=
'z') || ('A' <= c && c <= 'Z')); private boolean digit_set = false; private boolean digit; /*@ private invariant
digit_set ==> digit == ('0' <= c && c <= '9'); private boolean uppercase_set = false; private boolean
uppercase; /*@ private invariant uppercase_set ==> (uppercase == ('A' <= c && c <= 'Z')); private
boolean lowercase_set = false; private boolean lowercase; /*@ private invariant lowercase_set ==>
(lowercase == ('a' <= c && 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() { return vowel; } /*@ private normal_behavior @ assignable alphabetic_set,
alphabetic; @ ensures alphabetic_set && \result == alphabetic; @ public normal_behavior @ ensures
\result == ('a' <= c && c <= 'z') || ('A' <= c && 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 == ('A' <= c && 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 == ('a' <= c && 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 == ('0' <= c && 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 <==> ('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'); */ private
/*@ spec_public */ void setAlphabetic() { alphabetic = (('a' <= c && c <= 'z') || ('A' <= c && c <= 'Z'));
alphabetic_set = true; } /*@ private normal_behavior @ assignable uppercase_set, uppercase; @
ensures uppercase_set; @ ensures uppercase <==> ('A' <= c && c <= 'Z'); */ private /*@
spec_public */ void setUppercase() { uppercase = ('A' <= c && c <= 'Z'); uppercase_set = true; } /*@
private normal_behavior @ assignable lowercase_set, lowercase; @ ensures lowercase_set; @
ensures lowercase <==> ('a' <= c && c <= 'z'); */ private /*@ spec_public */ void setLowercase() {
lowercase = ('a' <= c && c <= 'z'); lowercase_set = true; } /*@ private normal_behavior @ assignable
digit_set, digit; @ ensures digit_set; @ ensures digit <==> ('0' <= c && c <= '9'); */ private /*@
```

```

spec_public @*/ void setDigit() { digit = ('0' <= c && 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.

The program is not correct based on the provided JML specification. The primary issue is with the implementation of the `isVowel()` method.

The JML specification for `isVowel()` states:

- It should return `true` if `c` is a vowel (either lowercase or uppercase) and `false` otherwise.

However, the implementation of `isVowel()` simply returns the value of `vowel` without first setting it. This means that the return value of `isVowel()` is not correctly determined based on the character `c`. The method `setVowel()` should be called within `isVowel()` to properly set the `vowel` flag based on the character `c`.

A counterexample would be any call to `isVowel()` without previously calling a method that sets `vowel_set` and `vowel` appropriately. In such a case, `isVowel()` will return the default value of `vowel` (which is `false`), regardless of whether `c` is a vowel or not.

For instance, if the constructor initializes `c` to `'a'`, and `isVowel()` is called without first calling `setVowel()`, it will incorrectly return `false` instead of `true`.



You did not verify the correctness of the "Alphabet" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



Upon re-evaluating the "Alphabet" program based on its JML specification, here's the assessment:

1. **Private Invariants:** The program defines several private invariants linking the character `c` with different character properties like being a vowel, alphabetic, digit, uppercase, and lowercase. These invariants correctly express the relationship between the `c` character and its properties.
2. **Constructor (`Alphabet(char c)`):** The constructor correctly sets the character `c` and ensures that the flags `vowel_set`, `alphabetic_set`, `digit_set`, `uppercase_set`, and `lowercase_set` are not set, which aligns with the specification.
3. **`isVowel()`, `isAlphabetic()`, `isUppercase()`, `isLowercase()`, `isDigit()` Methods:**
  - These methods should return true or false based on whether `c` is a vowel, alphabetic, uppercase, lowercase, or a digit, respectively.
  - However, they return the values of `vowel`, `alphabetic`, `uppercase`, `lowercase`, and `digit` directly without ensuring these variables are set according to the character `c` at the time of the method call.
  - The methods `setVowel()`, `setAlphabetic()`, `setUppercase()`, `setLowercase()`, and `setDigit()` correctly determine the properties of `c` and set the respective flags.
  - But these setter methods are not called in the corresponding getter methods, which means the returned values might not represent the current state of `c`.
4. **`driver(int op)` Method:** This method aims to test the character properties based on the operation specified. It appears to correctly call the relevant methods based on the operation code and sets the results in a boolean array.

Given this analysis, the program is not always correct based on the JML specification because the getter methods like `isVowel()`, `isAlphabetic()`, `isUppercase()`, `isLowercase()`, and `isDigit()` do not ensure that the properties of `c` are correctly evaluated at the time of their invocation. They should call their respective setter methods (like `setVowel()`) to ensure the private invariants are upheld.

In conclusion, the answer is NO, the program is not always correct based on the JML 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 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 == -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 withdraw(int amount) { if (isValid(amount)) { if (isValid(balance, amount)) { balance = balance -
amount; previousTransaction = -amount; } } } //@ 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); //@ requires isValid(balance, 50); //@
assignable balance, previousTransaction; //@ ensures balance == \old (balance) - 50; //@ ensures
previousTransaction == -50; //@ also //@ requires isValid(amount); //@ requires !isValid(balance,
amount); //@ requires !isValid(balance, 50); //@ assignable balance, previousTransaction; //@
ensures balance == 0; //@ ensures previousTransaction == \old (-balance); //@ also //@ requires
!isValid(amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction == \old
(previousTransaction); void checkWithdrawal(int amount) { if (isValid(amount)) { if (isValid(balance,
amount)) { balance = balance - amount; previousTransaction = -amount; } else { int
notEnoughMoneyPenalty; notEnoughMoneyPenalty = 50; int _balance; _balance = balance -
notEnoughMoneyPenalty; if (0 <= _balance) { balance = _balance; previousTransaction = -
notEnoughMoneyPenalty; } else { previousTransaction = -balance; balance = 0; } } } } //@ 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
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 isValid(_amount); //@
requires _amount + balance <= Integer.MAX_VALUE; //@ 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
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)*2; //@ 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)*2; //@ requires isValid(_amount); //@ requires !isValid(balance, _amount); //@ ensures
balance == \old (balance); //@ ensures previousTransaction == \old (previousTransaction); //@ also
//@ old int _amount = amount - (amount/100)*2; //@ requires !isValid(_amount); //@ ensures balance
== \old (balance); //@ ensures previousTransaction == \old (previousTransaction); void
withdrawByCashBack(int amount) { int cashback; cashback = (amount/100)*2; amount = amount -
cashback; if (isValid(amount)) { if (isValid(balance, amount)) { balance = balance - amount;
previousTransaction = -amount; } } } //@ old int ATMpenalty = 4; //@ requires amount + ATMpenalty <=
Integer.MAX_VALUE; //@ { } //@ requires isValid(amount); //@ requires isValid(balance, (amount +
ATMpenalty)); //@ assignable balance, previousTransaction; //@ ensures balance == \old (balance) -
(amount + ATMpenalty); //@ ensures previousTransaction == -(amount + ATMpenalty); //@ also //@
requires isValid(amount); //@ requires !isValid(balance, (amount + ATMpenalty)); //@ ensures balance
== \old (balance); //@ ensures previousTransaction == \old (previousTransaction); //@ also //@
requires !isValid(amount); //@ ensures balance == \old (balance); //@ ensures previousTransaction ==
\old (previousTransaction); //@ } void ATMWithdraw(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; _interest = balance/100; interest = _interest*2; } else if (balance <= 300000) { int _interest;
_interest = balance/100; interest = _interest*3; } else { int _interest; _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) - _amount; ensures previousTransaction == -_amount; 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; } }

```



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The primary issue lies within the second constructor `BankAccount(int currentBalance)`.

The JML specification for this constructor states:

- If `currentBalance` is less than or equal to 0, then `balance` should be set to 0.
- If `currentBalance` is greater than 0, then `balance` should be set to `currentBalance`.

However, the implementation of this constructor does the opposite:

- When `currentBalance` is greater than 0, it sets `balance` to 0.
- When `currentBalance` is less than or equal to 0, it sets `balance` to `currentBalance`.

A counterexample is creating a `BankAccount` object with a positive `currentBalance`, say 100. According to the specification, the `balance` should be set to 100, but due to the incorrect implementation, `balance` will be set to 0.



You:

You did not verify the correctness of the "BankAccount" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



Upon re-evaluation and focusing specifically on the JML specifications, let's examine the correctness of the "BankAccount" program:

1. **Invariants:** The program maintains the invariant  $0 \leq \text{balance}$ , ensuring the balance is always non-negative.
2. **First Constructor `BankAccount()`:**
  - Correctly initializes `balance` and `previousTransaction` to 0, aligning with the specification.
3. **Second Constructor `BankAccount(int currentBalance)`:**
  - Specification: If `currentBalance`  $\leq 0$ , `balance` should be 0; if `currentBalance`  $> 0$ , `balance` should be `currentBalance`.
  - Implementation: It incorrectly sets `balance` to 0 when `currentBalance`  $> 0$  and sets `balance` to `currentBalance` when `currentBalance`  $\leq 0$ . This is the opposite of the specified behavior.
4. **Third Constructor `BankAccount(int currentBalance, int _previousTransaction)`:**
  - Appears to be correctly implemented based on the JML specification.
5. **Methods `getBalance()`, `getPreviousTransaction()`, `isValid(int _amount)`, `isValid(int _balance, int _amount)`:**
  - Appear to be correctly implemented based on their JML specifications.
6. **Method `deposit(int amount)`:**
  - Appears to be correctly implemented based on the JML specification.
7. **Method `withdraw(int amount)`:**
  - Appears to be correctly implemented based on the JML specification.
8. **Method `checkWithdrawal(int amount)`:**
  - Appears to be correctly implemented based on the JML specification.
9. **Methods `foreignTransfer(int amount)`, `foreignDeposit(int amount)`, `withdrawByCashBack(int amount)`, `ATMWithdraw(int amount)`, `interestAfterYear()`:**
  - Appear to be correctly implemented based on their JML specifications.
10. **Method `menu(int option, int amount)`:**
  - Based on the provided JML specification, it appears to correctly handle different options and ensure the postconditions are met.

Given this analysis, the primary issue in the "BankAccount" program lies in the implementation of the second constructor `BankAccount(int currentBalance)`, which does not adhere to the JML specification. In all other aspects, the program appears to align with the specified behavior.

In conclusion, the answer is NO, the program is not always correct based on the JML specification due to the incorrect implementation of the second constructor.



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 BinarySearch { //@ requires \forall int j; 0 <= j && j < arr.length; \forall int i; 0 <= i && i < j; arr[i] <= arr[j]; //@ ensures \result == -1 <==> (\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; } }`



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The main issue is in the condition of the `if` statement at the beginning of the `Binary` method.

The specification for the `Binary` method states:

- The method should return `-1` if and only if the key is not present in the array (`\forall int i; 0 <= i && i < arr.length; arr[i] != key`).
- If the key is found, it should return the index of the key in the array (`0 <= \result && \result < arr.length ==> arr[\result] == key`).

However, the implementation has a flaw:

- The condition `if (arr.length != 0)` is incorrect. It should be `if (arr.length == 0)`. As written, the method will immediately return `-1` if the array is not empty, without performing the binary search.

A counterexample would be calling `Binary` with a non-empty array and a key that is present in the array. For example, calling `Binary(new int[] {1, 2, 3, 4, 5}, 3)` should return the index of 3 in the array (which is 2), but due to the incorrect `if` condition, it will return `-1`.



You:

You did not verify the correctness of the "BinarySearch" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The main issue is in the logic of the Binary method, specifically the initial if statement's condition.

The JML specification states that the method should return -1 if and only if the key is not present in the array. It also specifies that if the key is found in the array, the method should return the index of the key.

However, the implementation has a significant flaw in the condition `if (arr.length != 0)`. This condition should be `if (arr.length == 0)` to return -1 when the array is empty. As it stands, the method will return -1 for any non-empty array, without performing the binary search.

A counterexample to this implementation is any non-empty sorted array and a key that is present in the array. For instance, if we call `Binary(new int[] {1, 2, 3, 4, 5}, 3)`, the method should return the index of 3 (which is 2), but due to the incorrect condition, it will incorrectly return -1.



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 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--) {
            // 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.

The program is not correct based on the provided JML specification. The issue is in the outer for loop of the `bubbleSort` method.

The outer loop is intended to iterate over the array, reducing the range each time. However, the loop decrement `i--` is incorrect and should be `i++`. With `i--`, the loop will either run infinitely or not run at all, depending on the initial value of `i`.

A counterexample is any non-empty array, such as `{3, 2, 1}`. The method should sort the array, but due to the incorrect loop decrement, it will either not execute the loop or enter an infinite loop, failing to sort the array as per the JML specification.

-----



You:

You did not verify the correctness of the "BubbleSort" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?

-----



Upon re-evaluating the "BubbleSort" program in relation to the provided JML specification, it appears that the program is NOT correct.

The primary issue is in the outer loop of the `bubbleSort` method, specifically the loop increment/decrement statement. The loop uses `i--` instead of `i++`. This error causes the loop to either not execute or to run infinitely, depending on the initial value of `i`.

The JML specification requires that the method should sort the array in non-decreasing order. However, with the incorrect loop control, the array won't be sorted correctly.

A counterexample can be given with any non-empty array, such as `{2, 1, 3}`. The method is expected to sort this array to `{1, 2, 3}`, but due to the incorrect loop control (`i--`), it will either enter an infinite loop or not sort the array at all, thus not satisfying 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 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; case '%': output = num1 % num2; break; default: return -1; } return output; } }

```



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the `calculate` method, specifically in the case for the addition operator ('+').

The JML specification states:

- If the operator is '+', the method should return `num1 + num2`.

However, the implementation for the '+' case incorrectly performs subtraction (`num1 - num2`) instead of addition.

A counterexample would be calling `calculate` with `num1 = 2`, `num2 = 3`, and `operator = '+'`. According to the specification, the method should return 5 (`2 + 3`), but due to the incorrect implementation, it will return -1 (`2 - 3`).



You:

You did not verify the correctness of the "calculate" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:



NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the `calculate` method for the case of the addition operator ('+').

The JML specification states:

- If the operator is '+', the method should return `num1 + num2`.

However, the implementation for the '+' case incorrectly performs subtraction (`num1 - num2`) instead of addition.

A counterexample would be calling `calculate` with `num1 = 2`, `num2 = 3`, and `operator = '+'`. According to the specification, the method should return 5 (`2 + 3`), but due to the incorrect implementation, it will return -1 (`2 - 3`).



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 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(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(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); } } @*/ }
```



ChatGPT:

NO.

The program is not correct based on the provided JML specification, specifically in the combination method.

The JML specification for `combination(int n, int r)` states:

- The method should return the value of  $n$  choose  $r$ , which is calculated as  $\text{fac\_spec.spec\_factorial}(n) / (\text{fac\_spec.spec\_factorial}(r) * \text{fac\_spec.spec\_factorial}(n-r))$ .

However, the implementation incorrectly calculates the combination as  $\text{fac.factorial}(n) / (\text{fac.factorial}(r) * \text{fac.factorial}(n+r))$ . The denominator should be  $\text{fac.factorial}(r) * \text{fac.factorial}(n-r)$  instead of  $\text{fac.factorial}(r) * \text{fac.factorial}(n+r)$ .

A counterexample would be calling `combination(5, 3)`. The correct result should be 10, calculated as  $5! / (3! * 2!)$ . However, due to the incorrect implementation, it will return a different value.



You:

You did not verify the correctness of the "CombinationPermutation " method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification, specifically in the combination method.

The JML specification for `combination(int n, int r)` states:

- The method should return the value of  $n$  choose  $r$ , which is calculated as  $\text{fac\_spec.spec\_factorial}(n) / (\text{fac\_spec.spec\_factorial}(r) * \text{fac\_spec.spec\_factorial}(n-r))$ .

However, the implementation incorrectly calculates the combination as  $\text{fac.factorial}(n) / (\text{fac.factorial}(r) * \text{fac.factorial}(n+r))$ . The denominator should be  $\text{fac.factorial}(r) * \text{fac.factorial}(n-r)$  instead of  $\text{fac.factorial}(r) * \text{fac.factorial}(n+r)$ .

A counterexample would be calling `combination(5, 3)`. The correct result should be 10, calculated as  $5! / (3! * 2!)$ . However, due to the incorrect implementation, it will return a different value.

-----



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) {/while (iEnd - k > 0) { a[k] = b[k]; k = k + 1 ; } } }`



ChatGPT:

NO.

The program is not correct based on the provided JML specification, particularly due to the loop condition in the CopyArray method.

The JML specification states:

- The method should copy elements from array b to array a in the range from iBegin to iEnd - 1.

However, the implementation of the CopyArray method uses the loop condition `while (iEnd - k >= 0)`. This condition will result in an off-by-one error, causing the method to attempt to access and copy the element at index iEnd, which is beyond the specified range.

The correct loop condition should be `while (iEnd - k > 0)`, ensuring the loop iterates only until k is less than iEnd, and not equal to iEnd.

A counterexample would be calling CopyArray with b and a as non-empty arrays of the same length, iBegin = 0, and iEnd = b.length - 1. The method should copy elements up to the index iEnd - 1, but due to the incorrect loop condition, it will attempt to access and copy the element at index iEnd, resulting in an `ArrayIndexOutOfBoundsException`.



You:

You did not verify the correctness of the "CopyArray" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the loop condition of the CopyArray method.

The JML specification for the CopyArray method states:

- The method should copy elements from array b to array a from index iBegin (inclusive) to iEnd (exclusive).

However, the loop condition `while (iEnd - k >= 0)` in the implementation will cause the loop to continue until k equals iEnd, thereby attempting to copy the element at index iEnd which is outside the specified range. The correct behavior should stop copying when k reaches iEnd.

A counterexample can be provided with `iBegin = 0` and `iEnd = length` of the array. The method will attempt to access and copy the element at the index equal to the length of the array, which will result in an `ArrayIndexOutOfBoundsException`. The correct loop condition should be `while (iEnd - k > 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 FIND_FIRST_IN_SORTED { //@ requires 0 <= arr.length && arr.length <= (Integer.MAX_VALUE/2)+1; /*@ requires (\forall int j; 0 <= j && j < arr.length; @ (\forall int i; 0 <= i && i < j; arr[i] <= arr[j])); @*/ //@ ensures \result < arr.length; //@ ensures 0 <= \result && \result < arr.length ==> arr[\result] == x && (\forall int i; 0 <= i && i < \result; arr[i] != x); //@ ensures \result == -1 ==> (\forall 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 (\forall int i; 0 <= i && i < lo; arr[i] < x); //@ maintaining (\forall int i; hi < i && i < arr.length; x <= arr[i]); while (lo <= hi) { //while (lo < hi) { int mid = (lo + hi) / 2; if (x == arr[mid] && (mid == 0 || x != arr[mid-1])) { return mid; } else if (x <= arr[mid]) { hi = mid; } else { lo = mid + 1; } } //@ assume (\forall int i; 0 <= i && i < arr.length; arr[i] != x); return -1; } }`



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the loop condition of the `find_first_in_sorted` method.

The JML specification states:

- The method should find the first occurrence of  $x$  in the sorted array `arr` or return `-1` if  $x$  is not present.

However, the loop condition `while (lo <= hi)` is incorrect. This condition can cause an infinite loop or `ArrayIndexOutOfBoundsException` when `lo` equals `hi`. The correct condition should be `while (lo < hi)` to avoid these issues.

A counterexample can be provided with a sorted array containing at least one element and a search element  $x$  that is not in the array. For example, calling `find_first_in_sorted(new int[]{1, 2, 3}, 4)` should return `-1`, but due to the incorrect loop condition, it might result in an infinite loop or an exception.



You:

You did not verify the correctness of the "FIND\_FIRST\_IN\_SORTED" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The primary issue lies in the loop condition and the update of the `hi` variable in the `find_first_in_sorted` method.

The JML specification requires that the method returns the index of the first occurrence of  $x$  in the sorted array `arr`, or `-1` if  $x$  is not present.

However, the implementation has a flaw in the condition `while (lo <= hi)` and the update of `hi` in the `else if` branch. The loop should continue while `lo < hi` instead of `lo <= hi` to avoid an off-by-one error that can cause an infinite loop or an `ArrayIndexOutOfBoundsException`. Also, when `x <= arr[mid]`, the update `hi = mid` should be `hi = mid - 1` to exclude the `mid` element already checked.

A counterexample would be calling `find_first_in_sorted` with an array where  $x$  is present and  $x$  is equal to the middle element of the array. In such a case, the method can incorrectly skip the first occurrence of  $x$  or enter an infinite loop. For instance, in an array `[1, 2, 3, 4, 4, 5]` and searching for  $x = 4$ , the method might fail to return the index 3, which is the first occurrence of 4.



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 (\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 (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 -1; } 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.

The program is not correct based on the provided JML specification, specifically in the binsearch method.

The JML specification for binsearch states:

- The method should perform a binary search for x between indices start (inclusive) and end (exclusive) in a sorted array arr.
- It should return the index of x if found within the specified range, or -1 if x is not present.

However, the implementation has a flaw in the condition `if (start != end)`. This condition should be `if (start == end)`. As it stands, the method will return -1 immediately if start and end are not equal, which is the opposite of the required behavior. The method is intended to return -1 when the search range is empty (i.e., when start equals end).

A counterexample would be calling `find_in_sorted` with a sorted array containing the element x within its range. For instance, calling `find_in_sorted(new int[]{1, 2, 3, 4, 5}, 3)` should return the index of 3 (which is 2), but due to the incorrect condition, it will incorrectly return -1.



You did not verify the correctness of the "FIND\_IN\_SORTED" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?





NO.

The program is not correct based on the provided JML specification. The issue lies in the `binsearch` method.

The JML specification for `binsearch` states:

- If `start == end`, the method should return `-1`, indicating that the element `x` is not found in the empty range.

However, the implementation of `binsearch` incorrectly checks if `(start != end)`, and returns `-1` when `start` is not equal to `end`. This is the opposite of what the specification requires. As a result, the method returns `-1` immediately unless `start` and `end` are equal, which is not the intended behavior for a binary search.

A counterexample would be any call to `binsearch` where `start` is not equal to `end` (which is typically the case in a binary search). For example, calling `binsearch(new int[]{1, 2, 3, 4, 5}, 3, 0, 5)` should start a binary search for `3` in the entire array, but due to the incorrect condition, it will return `-1` instead of the correct index.

-----



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 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.

The program is not correct based on the provided JML specification, specifically in the `factorial` method.

The JML specification for `factorial` states:

- The method should return the factorial of  $n$ .

However, the implementation has a logical error in the `if` statement: `if (n != 0)`. This condition causes the method to return 1 (the initial value of `fact`) for any  $n$  that is not zero. The factorial of any number other than zero is not necessarily 1.

A counterexample would be calling `factorial` with  $n = 5$ . The correct result should be 120 (5!), but due to the incorrect `if` condition, it will return 1.



You:

You did not verify the correctness of the "Factorial" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue is in the implementation of the `factorial` method.

The JML specification for the `factorial` method states:

- The method should return the factorial of  $n$ .

However, in the implementation, there is a logical error in the `if` statement. The method returns 1 immediately if  $n$  is not zero (`if (n != 0)`), which is incorrect. The factorial of a number greater than 0 is not 1, except for the special case where  $n$  is 1.

A counterexample would be calling `factorial` with  $n = 3$ . According to the factorial definition, the result should be  $3 * 2 * 1 = 6$ , but due to the incorrect `if` condition, the method will incorrectly return 1.



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 (\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]; } }
```



NO.

The program is not correct based on the provided JML specification. The issue lies in the constructor `Fibonacci(int size)`.

The JML specification for this constructor states:

- It ensures that the first two elements of the `fib` array are 0 and 1, respectively.
- It also ensures that for any `i` where  $2 \leq i < \text{fib.length}$ , `fib[i]` should be initialized to 0.

However, in the implementation, after initializing the first two elements of the `fib` array, the constructor does not initialize the remaining elements of the array to 0. As a result, these elements will have the default value for long in Java, which is 0, but this is not explicitly ensured by the constructor as required by the specification.

A counterexample would be creating a `Fibonacci` object with `size > 2`. For example, `new Fibonacci(5)` should initialize the `fib` array as `{0, 1, 0, 0, 0}`, but the constructor only explicitly sets the first two elements, and the rest are implicitly set to 0 by Java's default initialization. This behavior does not fully comply with the "ensures" clause of the specification.



You did not verify the correctness of the "Fibonacci" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



NO.

The program is not correct based on the provided JML specification. The issue lies in the constructor `Fibonacci(int size)`.

The JML specification for this constructor states:

- It requires `2 <= size && size <= 93`.
- It ensures that `fib[0] == 0 && fib[1] == 1` and all other elements of `fib` for `2 <= i < fib.length` are initialized to 0.

However, in the implementation, the conditional check `if (2 < size && size <= 93)` is incorrect. The correct condition should be `if (2 <= size && size <= 93)`. With the current implementation, when the size is exactly 2, the constructor will throw an `IllegalArgumentException` instead of creating a `fib` array of size 2, which is contrary to the specification.

A counterexample would be calling the constructor with `size = 2`. According to the specification, this should create a `fib` array with two elements, both initialized correctly. However, due to the incorrect conditional check, it will throw an exception instead.



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 && (\forall int i; 0 <= i && i < \result; x[i] != 0); //@ ensures \result == -1 ==> (\forall 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 (\forall 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.

The program is not correct based on the provided JML specification. The issue lies in the condition of the `if` statement in the `FindFirstZero` method.

The JML specification states:

- If `x.length` is 0, the method should return -1.
- Otherwise, it should find the first occurrence of 0 in the array `x` and return its index or -1 if there is no 0.

However, in the implementation, the condition `if (x.length != 0)` is incorrect. It should be `if (x.length == 0)`. As currently implemented, the method will immediately return -1 if the array is not

empty, without searching for a 0.

A counterexample would be calling `FindFirstZero` with any non-empty array containing a 0, such as {1, 2, 0, 3, 4}. The method should return the index of the first 0 (which is 2), but due to the incorrect `if` condition, it will return -1.



You:

You did not verify the correctness of the "FindFirstZero " method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The main issue lies in the `FindFirstZero` method's conditional check and logic.

The JML specification for `FindFirstZero` states:

- If `x.length == 0`, the method should return -1.
- If there is a 0 in the array `x`, the method should return the index of the first occurrence of 0.
- If there is no 0 in the array, the method should return -1.

However, the implementation has a flaw in the condition `if (x.length != 0)`. This condition is reversed; it should be `if (x.length == 0)`. As a result, the method returns -1 for any non-empty array without checking for the first occurrence of 0.

A counterexample would be any non-empty array containing at least one 0, such as {1, 2, 0, 4, 5}. The method should return the index of the first 0 (which is 2 in this case), but due to the incorrect condition, it will return -1.



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(); } //@ assignable this.key; //@ ensures this.key == key; void
```

```

setKey(int key) { this.key = key; } // @ ensures \result == this.key; /* @ pure @ */ int getKey() { return
this.key; } // @ 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 && @ (\forall int i;
\result < i && i < arr.length; arr[i] != key)); @ ensures \result == -1 ==> (\forall int i; 0 <= i && i <
arr.length; arr[i] != key); @ /* @ pure @ */ int findLast() { int index = size() - 1; // @ maintaining -1 <=
index && index < arr.length; // @ maintaining (\forall 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 && @ (\forall int i; 0 <= i && i < \result; arr[i] != key)); @
ensures \result == -1 ==> (\forall int i; 0 <= i && i < arr.length; arr[i] != key); @ /* @ pure @ */ int
findFirst() { // @ maintaining 0 <= index && index <= arr.length; // @ maintaining (\forall 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.

The program is not correct based on the provided JML specification, specifically in the second constructor of the FindInArray class.

The JML specification for the second constructor FindInArray(int inputArr[], int key) states:

- It ensures that this.key is set to key.

However, in the implementation, the constructor does not set this.key to the provided key parameter. It only clones the inputArr to arr and does not assign the key parameter to this.key.

A counterexample would be creating an instance of FindInArray using this constructor with a non-zero key, such as new FindInArray(new int[]{1, 2, 3}, 5). According to the specification, this.key should be set to 5, but due to the missing assignment in the constructor, this.key will remain at its default value (0 in this case).



You:

You did not verify the correctness of the "FindInArray" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:



NO.

The program is not correct based on the provided JML specification. The issue lies in the second constructor `FindInArray(int inputArr[], int key)`.

The JML specification for this constructor states:

- It ensures `this.key == key`.
- It ensures that all elements of `inputArr` are copied to `arr`.

However, the implementation of the constructor does not set `this.key` to the provided key parameter. It only clones `inputArr` into `arr` but fails to assign the key parameter to `this.key`.

A counterexample would be creating a `FindInArray` object using this constructor with any array and a non-default key, such as `new FindInArray(new int[]{1, 2, 3}, 5)`. According to the specification, `this.key` should be set to 5. However, due to the missing assignment in the constructor, `this.key` will not be set to 5, resulting in a violation of the 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 GCD { /*@ public normal_behavior @ requires d != 0; @ ensures \result == n%d; @ pure function @*/ public static int div(int n, int d) { return n*d; } /*@ 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 (\forallall 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 (\forallall 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.

The program is not correct based on the provided JML specification. The primary issue lies in the `div` method.

The JML specification for the `div` method states:

- It should return the division remainder of `n` divided by `d` (`\result == n % d`).

However, the implementation of `div` incorrectly performs multiplication (`return n * d`) instead of division remainder.

A counterexample would be calling `div` with any values of `n` and `d` where `d` is not 0. For example, `div(5, 2)` should return 1 (the remainder of `5 / 2`), but due to the incorrect implementation, it will return 10 (the result of `5 * 2`). This error significantly impacts the correctness of the `gcd` method, which relies on the `div` method for its calculations.

-----



You:

You did not verify the correctness of the "GCD" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?

-----



ChatGPT:

NO.

The program is not correct based on the provided JML specification, primarily due to the implementation of the `div` method.

The JML specification for the `div` method states:

- It should return the division remainder of `n` divided by `d` (i.e., `n % d`).

However, the implementation of `div` is incorrect, as it returns the product of `n` and `d` (`return n * d`;) instead of the remainder of `n` divided by `d`. This error in the `div` method affects the correctness of the `gcd` method, which relies on `div` for its calculations.

A counterexample would be any call to the `gcd` method where both `num1` and `num2` are non-zero. The `gcd` method is supposed to calculate the greatest common divisor, but due to the incorrect implementation of `div`, it cannot correctly determine when a number divides both `num1` and `num2` without a remainder. For example, `gcd(8, 12)` should return 4, but due to the erroneous `div` implementation, it would not return the correct result.

-----



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; // 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.

The program is not correct based on the provided JML specification. The issue lies in the initial conditional check of the Inverse method.

The JML specification states:

- The method should return true if and only if arrays x and y are inverses of each other, meaning x.length should equal y.length, and for all i, x[i] should equal y[x.length - 1 - i].
- If x.length is not equal to y.length, the method should return false.

However, the implementation has a mistake in the condition `if (x.length == y.length) return false;`. This condition is the opposite of what is required. It should be `if (x.length != y.length) return false;`. As it stands, the method will return false immediately if the lengths of x and y are equal, without checking if they are inverses of each other.

A counterexample would be calling Inverse with two arrays that are inverses of each other, such as x = {1, 2, 3} and y = {3, 2, 1}. The method should return true, but due to the incorrect condition, it will return false.



You:

You did not verify the correctness of the "Inverse" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



NO.

The program is not correct based on the provided JML specification. The issue lies in the initial conditional check of the Inverse method.

The JML specification for the Inverse method states:

- The method should return true if x and y are inverses of each other. This means `x.length` should equal `y.length`, and for each element `i` in `x`, `x[i]` should equal `y[x.length - 1 - i]`.
- The method should return false if `x.length` is not equal to `y.length` or if there is at least one element in `x` that does not match the corresponding inverse element in `y`.

However, the implementation incorrectly checks `if (x.length == y.length) return false;`. This condition should be `if (x.length != y.length) return false;` in order to match the JML specification.

A counterexample would be any call to Inverse with two arrays of the same length but not inverse to each other. For instance, `Inverse(new int[]{1, 2, 3}, new int[]{1, 2, 3})` should return false as these arrays are not inverses of each other. However, due to the incorrect condition, the method will proceed to the while loop and may incorrectly return true.



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.

The program is not correct based on the provided JML specification, primarily due to the implementation of the `div` method.

The JML specification for the `div` method states:

- It should return the division remainder of  $n$  divided by  $d$  (i.e.,  $n \% d$ ).

However, the implementation of `div` returns the product of  $n$  and  $d$  (`return n * d;`), which is incorrect.

This incorrect implementation of `div` impacts the correctness of the `lcm` (Least Common Multiple) method, which relies on `div` for its calculations.

A counterexample would be any call to the `lcm` method with non-zero `num1` and `num2`. The method is supposed to calculate the least common multiple, but due to the incorrect implementation of `div`, it cannot correctly determine the least common multiple. For example, `lcm(6, 8)` should return 24, but due to the erroneous `div` implementation, it would not return the correct result.



You:

You did not verify the correctness of the "LCM" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification, primarily due to the implementation of the `div` method, which subsequently affects the correctness of the `lcm` method.

The JML specification for the `div` method states:

- It should return the division remainder of  $n$  divided by  $d$  (i.e.,  $n \% d$ ).

However, the implementation of `div` incorrectly performs multiplication (`return n * d;`) instead of division remainder. This error significantly impacts the correctness of the `lcm` method, which relies on `div` for its calculations.

A counterexample would be any call to the `lcm` method where both `num1` and `num2` are non-zero. The `lcm` method is supposed to calculate the least common multiple, but due to the incorrect implementation of `div`, it cannot correctly determine when a number is divisible by both `num1` and

num2. For example, `lcm(6, 8)` should return 24, but due to the erroneous `div` implementation, it would not return the correct result.



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 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; } }`



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the `isLeapYear` method.

The JML specification for `isLeapYear` states:

- A year is a leap year if it is divisible by 4 but not by 100, or if it is divisible by 400.

However, the implementation incorrectly checks `if (year * 4 == 0)` instead of `if (year % 4 == 0)`. The multiplication check will almost always result in a non-zero value, leading to incorrect evaluations of leap years.

A counterexample would be any leap year, such as the year 2020. According to the leap year rules, `isLeapYear(2020)` should return `true`, but due to the incorrect multiplication condition, it will incorrectly return `false`.



You:

You did not verify the correctness of the "isLeapYear" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?





NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the `isLeapYear` method.

The JML specification for `isLeapYear` states:

- The method should return true for a leap year, which is a year divisible by 4 but not 100, or divisible by 400.

However, the implementation incorrectly checks `if (year * 4 == 0)` instead of `if (year % 4 == 0)`. The `*` operator is used for multiplication, not for checking divisibility. The correct operator for checking divisibility is `%` (modulus).

A counterexample would be any year that is a multiple of 4. For example, calling `isLeapYear(2024)` should return true as 2024 is a leap year, but due to the incorrect condition (`year * 4 == 0`), it will return false.



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 <==> (\forallall 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 (\forallall 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; } }
```



NO.

The program is not correct based on the provided JML specification. The issue lies in the for-loop condition within the `linearSearch` method.

The JML specification for `linearSearch` states:

- It should return the index of search in array if it exists, otherwise -1.

However, the loop condition for `(c = 0; c <= array.length; c++)` will cause an `ArrayIndexOutOfBoundsException`. The condition should be `c < array.length` to ensure that the

loop does not attempt to access an index outside the bounds of the array.

A counterexample would be any call to `linearSearch` with a non-empty array and a search value not present in the array. For example, `linearSearch(5, new int[]{1, 2, 3, 4})` should return -1, but due to the incorrect loop condition, it will throw an `ArrayIndexOutOfBoundsException` when trying to access `array[4]`.



You did not verify the correctness of the "LinearSearch " method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



NO.

The program is not correct based on the provided JML specification. The issue is in the for loop condition within the `linearSearch` method.

The JML specification for `linearSearch` states:

- It should return the index of search in array if it exists, otherwise -1.

However, the loop condition for `(c = 0; c <= array.length; c++)` is incorrect. This condition allows `c` to reach the value `array.length`, which is an out-of-bounds index for the array. The correct condition should be `c < array.length`.

A counterexample would be any call to `linearSearch` with a search value not present in the array. For instance, calling `linearSearch(5, new int[]{1, 2, 3, 4})` should return -1. However, due to the incorrect loop condition, it will attempt to access `array[array.length]` (in this case, `array[4]`), leading to an `ArrayIndexOutOfBoundsException`.



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



NO.

The program is not correct based on the provided JML specification, specifically in the `isEven` method.

The JML specification for `isEven` states:

- The method should return true if  $x$  is even (i.e.,  $x \% 2 == 0$ ).

However, the implementation incorrectly checks `return x * 2 == 0`; (multiplication by 2) instead of `return x \% 2 == 0`; (modulo 2). This condition will almost always result in false, except when  $x$  is 0.

A counterexample would be any even number other than 0. For instance, calling `isEven(2)` should return true as 2 is an even number, but due to the incorrect implementation, it will return false.

-----



You:

You did not verify the correctness of the "OddEven" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?

-----



NO.

The program is not correct based on the provided JML specification due to the implementation of the `isEven` method.

The JML specification for `isEven` states:

- The method should return true if and only if  $x$  is an even number (i.e.,  $x \% 2 == 0$ ).

However, the implementation of `isEven` checks `x * 2 == 0`, which is a multiplication operation, not a modulo operation. It should use `x \% 2 == 0` to correctly check for even numbers.

A counterexample would be calling `isEven` with any even number other than 0, such as `isEven(2)`. According to the specification, the method should return true, but due to the incorrect implementation (`x * 2 == 0`), it will return false.



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
squarePerimeter(short x) { long
squarePerimeter = 4 / (long)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.

The program is not correct based on the provided JML specification due to the implementation of the first `Perimeter` method which calculates the perimeter of a square.

The JML specification for this method states:

- It requires  $0 < x \ \&\& \ x \leq \text{Short.MAX\_VALUE}$ .
- It ensures  $\text{\result} == 4 * x$ .

However, the implementation of this method is incorrect: `long squarePerimeter = 4 / (long)x;` performs division instead of multiplication. The correct implementation should be `long squarePerimeter = 4 * (long)x;`.

A counterexample would be calling the `Perimeter` method for a square with any positive short value for `x`, such as `Perimeter((short)5)`. According to the specification, the method should return 20 ( $4 * 5$ ), but due to the incorrect implementation (division instead of multiplication), it will return an incorrect result.



You:

You did not verify the correctness of the "Perimeter" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification, specifically due to the implementation of the first Perimeter method for calculating the perimeter of a square.

The JML specification for this method states:

- It ensures `\result == 4*x`.

However, the implementation incorrectly calculates the perimeter with `long squarePerimeter = 4 / (long)x`; This line performs division instead of multiplication. The correct implementation should be `long squarePerimeter = 4 * (long)x`; to adhere to the specification.

A counterexample would be calling the Perimeter method for a square with any positive short value for x, such as `Perimeter((short)5)`. According to the specification, the method should return 20 ( $4 * 5$ ). However, due to the incorrect implementation, it will return an incorrect result (division instead of multiplication).



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.

The program is not correct based on the provided JML specification, due to the implementation of the div method.

The JML specification for the `div` method states:

- It should return the division remainder of  $n$  divided by  $d$  (i.e.,  $n \% d$ ).

However, the implementation of `div` returns the product of  $n$  and  $d$  (`return n * d;`), which is incorrect for calculating division remainders.

This incorrect implementation of `div` affects the correctness of the `isPrime` method, which relies on `div` to check if a number  $a$  is divisible by any number from 2 to  $a/2$ .

A counterexample would be any call to `isPrime` with a prime number. For instance, calling `isPrime(5)` should return `true` as 5 is a prime number. However, due to the erroneous `div` implementation, it would return `false` since `div` does not correctly calculate the remainder.



You:

You did not verify the correctness of the "PrimeCheck " method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification, due to the implementation of the `div` method.

The JML specification for the `div` method states:

- It should return the division remainder of  $n$  divided by  $d$  (i.e.,  $n \% d$ ).

However, the implementation of `div` incorrectly performs multiplication (`return n * d;`) instead of division remainder.

This error significantly impacts the correctness of the `isPrime` method, which relies on `div` for its calculations. The `isPrime` method is supposed to determine if a number is prime by checking for any divisors between 2 and  $a/2$ . However, due to the incorrect implementation of `div`, it cannot correctly determine if a number divides  $a$  without a remainder.

A counterexample would be any call to the `isPrime` method with a prime number. For instance, calling `isPrime(5)` should return `true` as 5 is a prime number. However, due to the erroneous `div` implementation, it would not return the correct result.



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.

The program is not correct based on the provided JML specification, particularly due to the implementation of the `div` method.

The JML specification for the `div` method states:

- The method should return true if  $n$  is divisible by  $d$  (i.e.,  $n \% d == 0$ ).

However, the implementation of `div` incorrectly checks `return n * d == 0`; (multiplication) instead of `n % d == 0`; (division remainder). This mistake causes the method to return true only when either  $n$  or  $d$  is zero, which is not the correct behavior for checking divisibility.

A counterexample would be any call to the `div` method with non-zero  $n$  and  $d$  where  $n$  is divisible by  $d$ . For example, `div(6, 3)` should return true as 6 is divisible by 3, but due to the incorrect implementation, it will return false. This error significantly impacts the correctness of the `primeList` method, which relies on `div` to check for prime numbers.





You:

You did not verify the correctness of the "PrimeNumbers" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification due to the implementation of the `div` method.

The JML specification for the `div` method states:

- It should return true if `n` is divisible by `d` (i.e., `n % d == 0`).

However, the implementation of `div` incorrectly checks `return n * d == 0`; (multiplication) instead of `n % d == 0`; (division remainder). This mistake causes the method to return true only when either `n` or `d` is zero, which is not the correct behavior for checking divisibility.

A counterexample would be any call to the `div` method with non-zero `n` and `d` where `n` is divisible by `d`. For instance, `div(6, 3)` should return true as 6 is divisible by 3, but due to the incorrect implementation, it will return false. This error significantly impacts the correctness of the `primeList` method, which relies on `div` to check for prime numbers.



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.

The program is not correct based on the provided JML specification. The primary issue is in the conditional check at the beginning of the `Smallest` method.

The JML specification for the `Smallest` method states:

- If `a.length` is 0, the method should return -1.

However, the implementation incorrectly checks `if (a.length != 0) return -1;`. This condition should be `if (a.length == 0) return -1;` to return -1 when the array is empty, as specified.

A counterexample would be calling `Smallest` with any non-empty array, such as `{1, 2, 3}`. According to the specification, the method should return the index of the smallest element (which is 0 in this case), but due to the incorrect `if` condition, it will return -1.



You:

You did not verify the correctness of the "Smallest" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the initial conditional check of the `Smallest` method.

The JML specification for `Smallest` states:

- If `a.length` is 0, the method should return -1.
- Otherwise, it should find the index of the smallest element in the array `a`.

However, the implementation incorrectly checks `if (a.length != 0) return -1;`. This condition should be `if (a.length == 0) return -1;`. As it stands, the method will return -1 immediately if the array is not empty, without searching for the smallest element.

A counterexample would be any non-empty array, such as `{2, 1, 3}`. According to the specification, the method should return the index of the smallest element (1 in this case), but due to the incorrect `if` condition, it will return -1.



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); } }
```



NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the `isPalindrome` method, specifically in the loop that constructs the reverse string.

The JML specification for `isPalindrome` states:

- It should return true if and only if the reverse of the string `str` is equal to `str`.

However, the loop condition in the `isPalindrome` method is `for (int i = length - 1; 0 < i; i--)`, which means the loop will not include the character at index 0 in the reverse string. The correct loop condition should be `for (int i = length - 1; 0 <= i; i--)` to ensure all characters, including the one at index 0, are included in the reverse string.

A counterexample would be any palindrome string of length greater than 1, such as "aba". The method should return true, but due to the incorrect loop condition, it will not correctly reverse the entire string and therefore will not recognize it as a palindrome.



You did not verify the correctness of the "StrPalindrome" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



NO.

The program is not correct based on the provided JML specification due to the implementation of the `isPalindrome` method.

The JML specification for `isPalindrome` states:

- The method should return true if and only if the string `str` is a palindrome, meaning `str` should be equal to reverse.

However, the loop in `isPalindrome` incorrectly constructs the reverse string. The loop condition is for `(int i = length - 1; 0 < i; i--)`, which omits the first character of `str` (at index 0) in reverse. The correct condition should be for `(int i = length - 1; 0 <= i; i--)` to include all characters of `str` in reverse.

A counterexample would be a single-character string, such as "a". According to the specification, the method should return true as a single-character string is a palindrome. However, due to the incorrect loop condition, reverse will be an empty string, and the method will incorrectly return false.

-----



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) { //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.

The program is not correct based on the provided JML specification. The primary issue lies in the implementation of the `div` method.

The JML specification for the `div` method states:

- It should return true if  $n$  is divisible by  $d$  (i.e.,  $n \% d == 0$ ).

However, the implementation of `div` incorrectly performs multiplication and comparison to zero (`return n * d == 0;`), which is not the correct logic for checking divisibility. This implementation will



always return false unless one of the arguments is 0, which is not the intended behavior for a divisibility check.

This incorrect implementation of the `div` method affects the correctness of other methods in the class that depend on it for checking divisibility conditions.



You:

You did not verify the correctness of the "StudentEnrollment" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



ChatGPT:

NO.

The program is not correct based on the provided JML specification, particularly due to the implementation of the `setTuitionBalance` method.

The JML specification for `setTuitionBalance` states:

- The method should allow setting `this.tuitionBalance` to `tuitionBalance` if `tuitionBalance` is less than or equal to `maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6)`.
- It should throw an `IllegalArgumentException` if `tuitionBalance` is greater than this value.

However, the implementation incorrectly checks `if (maxTuitionBalance <= tuitionBalance)` instead of `if (maxTuitionBalance < tuitionBalance)`. This condition means that the method will throw an exception even when `tuitionBalance` is exactly equal to `maxTuitionBalance`, which contradicts the specification.

A counterexample would be calling `setTuitionBalance` with a `tuitionBalance` value equal to `maxSemesterCredits * costPerCredit + maxSemesterCredits * ((costPerCredit/100)*6)`. According to the specification, this should be allowed, but due to the incorrect condition in the implementation, it will throw an `IllegalArgumentException`.



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 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(); } }

```



ChatGPT:

NO.

The program is not correct based on the provided JML specification, particularly due to the implementation of the `setSecond` method.

The JML specification for `setSecond` states:

- It requires  $0 \leq s \ \&\& \ s < 60$ .
- It ensures `this.second == s`.

However, the implementation of `setSecond` has an incorrect conditional check: `if (s <= 0 || 60 <= s)`. The correct condition should be `if (s < 0 || 60 <= s)`. As currently implemented, the method will throw an `IllegalArgumentException` for  $s = 0$ , which is a valid value for seconds.

A counterexample would be calling `setSecond` with  $s = 0$ . According to the specification, this should set `this.second` to 0, but due to the incorrect conditional check, it will throw an `IllegalArgumentException`.



You:

You did not verify the correctness of the "Time" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?



NO.

The program is not correct based on the provided JML specification due to the implementation of the `setSecond` method.

The JML specification for `setSecond` states:

- It requires  $0 \leq s \ \&\& \ s < 60$ .
- It ensures `this.second == s`.

However, the implementation of `setSecond` has an incorrect conditional check: `if (s <= 0 || 60 <= s)`. The correct condition should be `if (s < 0 || 60 <= s)`. As it stands, the method will throw an `IllegalArgumentException` for  $s = 0$ , which is a valid value for seconds according to the specification.

A counterexample would be calling `setSecond` with  $s = 0$ . According to the specification, this should set `this.second` to 0, but due to the incorrect conditional check, it will throw an `IllegalArgumentException` instead of setting the second to 0.



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.

The program is not correct based on the provided JML specification. The issue lies in the for-loop that performs the transposition of the matrix.

The JML specification for transposeMat states:

- The method should return a transposed matrix of matrix.

However, the implementation has a loop boundary issue in the outer for-loop: `for (int c = 0; c <= n; c++)`. The condition should be `c < n` to avoid an `ArrayIndexOutOfBoundsException`. As it stands, when `c` equals `n`, `transpose[c]` will cause an out-of-bounds access.

A counterexample would be any non-empty matrix, such as a 2x2 matrix `{{1, 2}, {3, 4}}`. According to the specification, the method should return the transposed matrix without any error, but due to the incorrect loop condition, it will throw an `ArrayIndexOutOfBoundsException` when attempting to access `transpose[n]`.

-----



You:

You did not verify the correctness of the "TransposeMatrix" method based on JML specification. However, the generated counterexamples are not valid based on JML specifications. Is the program always correct based on the JML specification?

-----



ChatGPT:

NO.

The program is not correct based on the provided JML specification. The issue lies in the implementation of the transposeMat method, specifically in the outer for-loop.

The JML specification for transposeMat states:

- The method should return a transposed matrix of matrix.

However, the implementation has a loop boundary issue in the outer for-loop: `for (int c = 0; c <= n; c++)`. The condition should be `c < n` to avoid an `ArrayIndexOutOfBoundsException`. As it stands, when `c` equals `n`, `transpose[c]` will cause an out-of-bounds access.

A counterexample would be any non-empty matrix, such as a 2x2 matrix `{{1, 2}, {3, 4}}`. According to the specification, the method should return the transposed matrix `{{1, 3}, {2, 4}}` without any error, but due to the incorrect loop condition, it will throw an `ArrayIndexOutOfBoundsException` when attempting to access `transpose[n]`, which is outside the bounds of the array.

-----