Which of the following are part of a program's state? Choose all that apply.

	Answer	Respondents	Percentage
✓	The values of all variables in the program.	14	25%
×	The sequence of instructions the program will execute in the future.	3	5%
✓	The current position of execution in the program.	14	25%
✓	Any output produced so far.	10	18%
✓	The input that has been provided but not yet processed.	14	25%
×	The source code of the program.	0	0%

Preconditions, postconditions, and loop invariants are statements about ...

	Answer	Respondents	Percentage
✓	program state, not process	11	73%
×	program process, not state	4	27%

statements about a snapshot

Which of the following are postconditions for the following method? Choose all that apply.

```
public int max(int a, int b) {
    return (a > b) ? a : b;
}
```

	Answer	Respondents	Percentage
×	a and b must be integers	7	18%
×	a >= 0 and b >= 0	3	8%
✓	The returned value must be either a or b.	13	34%
×	The returned value is the larger of a and b.	11	29%
×	The method should return -1 if a and b are equal.	3	8%
×	The returned value must always be positive.	1	3%

postconditions are about the result

true, but statements that the result is the correct answer are often not easily checkable

a valid postcondition if that is desired behavior – though not in conjunction with the previous two

a valid postcondition if there is also a precondition a,b > 0

Which of the following are valid preconditions, postconditions, and loop invariants for the code below? Choose all that apply.

```
public int sumPositiveNumbers(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > 0) {
            sum += arr[i];
        }
    }
    return sum;
}
```

	Answer	Respondents	Percentage
✓	precondition: arr != null	12	14%
×	precondition: all numbers in the array must be positive	7	8%
×	precondition: arr.length > 0	7	8%
×	postcondition: every element of the array is processed once	8	10%
✓	postcondition: the returned value is the sum of the positive numbers in the array	15	18%

no such requirement for correct functioning

no such requirement for correct functioning – returns 0 if `arr.length == 0`, which is a valid sum of no numbers

true, but is a statement about process rather than state

valid postcondition, though difficult to check

Which of the following are valid preconditions, postconditions, and loop invariants for the code below? Choose all that apply.

```
public int sumPositiveNumbers(int[] arr) {
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] > 0) {
            sum += arr[i];
        }
    }
    return sum;
}
```

×	loop invariant: the loop iterates through the array one element at a time from left to right	6	7%	about process, not state
✓	loop invariant: sum contains the sum of the positive numbers in the first i slots of arr	7	8%	
×	loop invariant: sum contains the sum of the positive numbers encountered so far	10	12%	valid type of statement (about state) but not precise enough for reasoning about correctness
×	loop invariant: if arr[i] > 0, then sum increases by arr[i] in this iteration	11	13%	about process, not state

Assertions

Assertions let you state a boolean condition that should be true at that point in the program.

- if it is, program execution continues normally
- if it isn't, the program terminates

Syntax:

```
assert condition;
assert condition : error-message;
```

- if the condition is true, nothing happens (program continues)
- if the condition is false, an exception is generated (with the optional error message)

Using Assertions

```
// suit is one of “spades”, “diamonds”, “hearts”,
//  “clubs”

if ( suit.equals(“spades”) ) {
  …
} else if ( suit.equals(“diamonds”) ) {
  …
} else if ( suit.equals(“hearts”) ) {
  …
} else {
  assert suit.equals(“clubs”);
  …
}
```


Using Assertions

```
// i % 3 must be 0, 1, or 2  i.e. i >= 0

if ( i % 3 == 0 ) {
  …
} else if ( i % 3 == 1 ) {
  …
} else {
  assert i % 3 == 2;
  …
}
```

- an alternative is `assert i >= 0` before the statement

Class Invariant

include only useful checks – class invariant should be true at beginning and end of each method, but with private instance variables, their values can't be changed between method calls

```
public class BankAccount {

  private double balance_;   // balance >= 0

  public BankAccount () {
    balance_ = 0;
    assert balance >= 0;
  }

  public void withdraw ( double amount ) {
    balance_ -= amount;
    assert balance >= 0;
  }

  public void deposit ( double amount ) {
    balance_ += amount;
    assert balance >= 0;
  }
}
```

identifying class invariant reveals need for precondition

Class Invariant / Data Structure Constraint

```
public class SortedArray {

  private int[] array_;   // in increasing order
  private int size_;

  public SortedArray ( int capacity ) {
    array_ = new int[capacity];
    size_ = 0;
    assert isSorted();
   }

  public void insert ( int elt ) {
    for ( int i = size_ ; i >= 0 ; i-- ) {
      if ( array_[i-1] > elt ) {
        array_[i] = array_[i-1];
      } else {
        array_[i] = elt;
        break;
      }
    }
    assert isSorted();
  }
  private boolean isSorted () { … }
}
```


Assertions

An advantage of assertions is that they can be turned on and off.

- can be left in production code without incurring a performance hit
  - checking assertion condition may be expensive
- can be turned on for testing and debugging

Note: assertions are disabled by default.

Enable for the whole program with the runtime argument

```
-ea
```

- in Eclipse, this is under Run Configurations “VM Arguments”
- can also selectively enable assertions for particular classes – see section 8.4.1 in the text

In which of the following circumstances is it most appropriate to use assertions? Choose all that apply.

	Answer	Respondents	Percentage
✓	To check for conditions that should never happen if the program is written correctly.	10	22%
×	To validate user input.	5	11%
✓	To verify that a private helper method receives a valid argument.	10	22%
×	To handle runtime errors that may occur due to invalid user input.	6	13%

assertions are for internal correctness checks – bad user input isn’t the program’s fault

In which of the following circumstances is it most appropriate to use assertions? Choose all that apply.

✕	To check for conditions that may change depending on external factors, such as opening a file that may not exist.	3	7%
✓	When debugging and verifying internal assumptions about program logic.	11	24%
✕	When validating method arguments in a public API.	0	0%

assertions are for internal correctness checks – external factors are not the program's fault