

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 3 : 28 Feb - 4 March — Checking and Testing_2, Immutability, List, Set, Map](#) / [Lecture Quiz #3](#)

Started on Thursday, 2 March 2023, 15:34

State Finished

Completed on Thursday, 2 March 2023, 16:13

Time taken 39 mins 16 secs

Grade 110.00 out of 150.00 (73.33%)

Question 1

Correct

Mark 10.00 out of 10.00

In the buggy Java code below, is the bug caught by static checking, dynamic checking, or not at all?

```
1. int n = 5;
2. if (n) {
3.     System.out.println("n is " + n);
4. }
```

Select one:

- a. static checking ✓
- b. dynamic checking
- c. no checking, resulting in wrong answer

Your answer is correct.

The correct answer is: static checking

Question 2

Correct

Mark 10.00 out of 10.00

In the buggy Java code below, is the bug caught by static checking, dynamic checking, or not at all?

```
1. int bigNum = 200000;           // bigNum is 200,000
2. bigNum = bigNum * bigNum;      // bigNum should be 40 billion
```

Select one:

- a. static checking
- b. dynamic checking
- c. no checking, resulting in wrong answer ✓

Your answer is correct.

The correct answer is: no checking, resulting in wrong answer

Question 3

Correct

Mark 10.00 out of 10.00

In the buggy Java code below, is the bug caught by static checking, dynamic checking, or not at all?

```
1. // the probability of an event is prob = 1/5 = 0.2
2. double prob = 1/5;
```

Select one:

- a. static checking
- b. dynamic checking
- c. no checking, resulting in wrong answer✓

Your answer is correct.

The correct answer is: no checking, resulting in wrong answer

Question 4

Correct

Mark 10.00 out of 10.00

In the buggy Java code below, is the bug caught by static checking, dynamic checking, or not at all?

```
1. int sum = 0;
2. int n = 0;
3. int average = sum/n;
```

Select one:

- a. static checking
- b. dynamic checking✓
- c. no checking, resulting in wrong answer

Your answer is correct.

The correct answer is: dynamic checking

Question 5

Incorrect

Mark 0.00 out of 10.00

In the buggy Java code below, is the bug caught by static checking, dynamic checking, or not at all?

1. `double sum = 7;`
2. `double n = 0;`
3. `double average = sum/n;`

Select one:

- a. static checking
- b. dynamic checking 
- c. no checking, resulting in wrong answer

Your answer is incorrect.

The correct answer is: no checking, resulting in wrong answer

Question 6

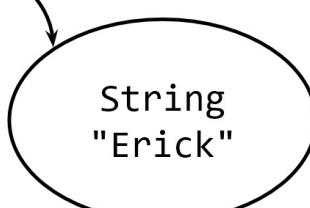
Correct

Mark 10.00 out of 10.00

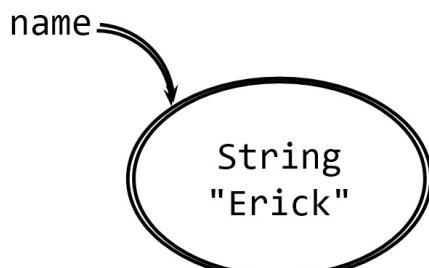
Which is the correct snapshot diagram for:

```
final String name = "Erick";
```

Select one:

- name → 
- name → 
- name → 
- name → 

Your answer is correct.



The correct answer is:

Question 7

Incorrect

Mark 0.00 out of 10.00

Choose the **incorrect** statement:

Select one:

- a. String is an immutable type.
- b. StringBuilder is a mutable type.
- c. final variable cannot be reassigned. 
- d. object pointed by final variable cannot be mutated.
- e. List is a mutable type.

Your answer is incorrect.

The correct answer is: object pointed by final variable cannot be mutated.

Question 8

Correct

Mark 20.00 out of 20.00

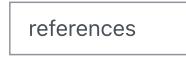
When you try to reassign a final variable, Java compiler will produce a compile error.

Therefore, final provides you



static checking

 for immutable



references

 .

Question 9

Incorrect

Mark 0.00 out of 10.00

Rewrite the variable declaration below using Lists instead of arrays:

1.  char[][] matrix;

Answer:  List<char> matrix 

The correct answer is: List<List<Character>> matrix ;

Question 10

Correct

Mark 10.00 out of 10.00

Given a code:

```
List<Integer> list1 = new ArrayList<>();  
list1.add(100);  
list1.add(200);  
final List<Integer> list2 = list1;  
list1.add(300);
```

If we add a line of code below:

```
list2 = list1;
```

choose the **correct** statement:

Select one:

- a. there will be an error, detected by static checking.✓
- b. there will be an error, detected by dynamic checking.
- c. there is no error.

Your answer is correct.

The correct answer is: there will be an error, detected by static checking.

Question 11

Correct

Mark 10.00 out of 10.00

Given a code:

```
List<Integer> list1 = new ArrayList<>();  
list1.add(100);  
list1.add(200);  
final List<Integer> list2 = list1;  
list1.add(300);
```

If we add a line of code below:

```
list1.set(1, 400);
```

choose the **correct** statement:

Select one:

- a. there will be an error, detected by static checking.
- b. there will be an error, detected by dynamic checking.
- c. there is no error.✓

Your answer is correct.

The correct answer is: there is no error.

Question 12

Correct

Mark 10.00 out of 10.00

Given a code:

```
List<Integer> list1 = new ArrayList<>();  
list1.add(100);  
list1.add(200);  
final List<Integer> list2 = list1;  
list1.add(300);
```

If we add a line of code below:

```
list2.set(1, 400);
```

choose the **correct** statement:

Select one:

- a. there will be an error, detected by static checking.
- b. there will be an error, detected by dynamic checking.
- c. there is no error.✓

Your answer is correct.

The correct answer is: there is no error.

Question 13

Incorrect

Mark 0.00 out of 10.00

Given a code:

```
List<Integer> list1 = new ArrayList<>();  
list1.add(100);  
list1.add(200);  
final List<Integer> list2 = list1;  
list1.add(300);
```

If we add a line of code below:

```
list2.set(3, 400);
```

choose the **correct** statement:

Select one:

- a. there will be an error, detected by static checking.
- b. there will be an error, detected by dynamic checking.
- c. there is no error.✗

Your answer is incorrect.

The correct answer is: there will be an error, detected by dynamic checking.

Question 14

Correct

Mark 10.00 out of 10.00

Create a map named hostel with integer keys and string values, to store room number and tenant name pairs.

Then, add a key-value pair for a tenant named Alice in room number 777.

Select one:



```
Map<Integer, String> hostel = new HashMap<>();  
  
hostel.add(777, "Alice");
```



```
Map<Integer, String> hostel = new HashMap<>();  
  
hostel.put(777, "Alice");
```



```
Map<String, Integer> hostel = new HashMap<>();  
  
hostel.add("Alice", 777);
```



```
Map<String, Integer> hostel = new HashMap<>();  
  
hostel.put("Alice", 777);
```



```
Map<String, int> hostel = new HashMap<>();  
  
hostel.add("Alice", 777);
```



```
Map<String, int> hostel = new HashMap<>();  
  
hostel.put("Alice", 777);
```

Your answer is correct.

The correct answer is:

```
Map<Integer, String> hostel = new HashMap<>();  
  
hostel.put(777, "Alice");
```

[◀ CPT204 2223 Lab 3 Video Recordings](#)

Jump to...

[Lab Exercise #3.1 Max Stretch ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#) / [Week 4 : 6-10 March — Coding Rules, Testing 3, Recursion](#) / [Lecture Quiz #4](#)

Started on Wednesday, 8 March 2023, 18:05

State Finished

Completed on Wednesday, 8 March 2023, 18:41

Time taken 36 mins 26 secs

Grade 120.00 out of 130.00 (92.31%)

Question 1

Correct

Mark 10.00 out of 10.00

Somebody wrote a **bad** code that **does not fail fast** (from the Lecture 3):

```

1. public static int dayOfYear(int month, int dayOfMonth, int year) {
2.     if (month == 2) {
3.         dayOfMonth += 31;
4.     } else if (month == 3) {
5.         dayOfMonth += 59;
6.     } else if (month == 4) {
7.         dayOfMonth += 90;
8.     } else if (month == 5) {
9.         dayOfMonth += 31 + 28 + 31 + 30;
10.    } else if (month == 6) {
11.        dayOfMonth += 31 + 28 + 31 + 30 + 31;
12.    } else if (month == 7) {
13.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30;
14.    } else if (month == 8) {
15.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31;
16.    } else if (month == 9) {
17.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
18.    } else if (month == 10) {
19.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
20.    } else if (month == 11) {
21.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
22.    } else if (month == 12) {
23.        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
24.    }
25.    return dayOfMonth;
26. }
```

Assume today is **January 3, 2019**,

which means that the correct *dayOfYear* for this date is 3,
since it's the third day of the year.

Now **another programmer** calls that method with arguments as follows:

```
1. dayOfYear(1, 3, 2019)
```

Choose the **correct** statement:

Select one:

- a. The programmer did not make a mistake. ✓
The method gave the right answer.
- b. The programmer made a mistake.
The method gave the right answer, luckily.
- c. The programmer made a mistake.
The method gave the wrong answer, quietly.
- d. The programmer made a mistake.
The method detected a static error.
- e. The programmer made a mistake.
The method detected a dynamic error.

Your answer is correct.

The correct answer is: The programmer did not make a mistake.

The method gave the right answer.

Question 2

Correct

Mark 10.00 out of 10.00

Now **another programmer** calls that method with arguments as follows:

1. `dayOfYear(0, 3, 2019)`

Choose the **correct** statement:

Select one:

- a. The programmer did not make a mistake.
The method gave the right answer.
- b. The programmer made a mistake. ✓
The method gave the right answer, luckily.
- c. The programmer made a mistake.
The method gave the wrong answer, quietly.
- d. The programmer made a mistake.
The method detected a static error.
- e. The programmer made a mistake.
The method detected a dynamic error.

Your answer is correct.

The correct answer is: The programmer made a mistake.

The method gave the right answer, luckily.

Question 3

Correct

Mark 10.00 out of 10.00

Now **another programmer** calls that method with arguments as follows:

1. `dayOfYear(3, 1, 2019)`

Choose the **correct** statement:

Select one:

- a. The programmer did not make a mistake.
The method gave the right answer.
- b. The programmer made a mistake.
The method gave the right answer, luckily.
- c. The programmer made a mistake. ✓
The method gave the wrong answer, quietly.
- d. The programmer made a mistake.
The method detected a static error.
- e. The programmer made a mistake.
The method detected a dynamic error.

Your answer is correct.

The correct answer is: The programmer made a mistake.

The method gave the wrong answer, quietly.

Question 4

Correct

Mark 10.00 out of 10.00

Now **another programmer** calls that method with arguments as follows:

1. `dayOfYear("January", 3, 2019)`

Choose the **correct** statement:

Select one:

- a. The programmer did not make a mistake.
The method gave the right answer.
- b. The programmer made a mistake.
The method gave the right answer, luckily.
- c. The programmer made a mistake.
The method gave the wrong answer, quietly.
- d. The programmer made a mistake. ✓
The method detected a static error.
- e. The programmer made a mistake.
The method detected a dynamic error.

Your answer is correct.

The correct answer is: The programmer made a mistake.

The method detected a static error.

Question 5

Correct

Mark 10.00 out of 10.00

Now **another programmer** calls that method with arguments as follows:

1. `dayOfYear(2019, 1, 3)`

Choose the **correct** statement:

Select one:

- a. The programmer did not make a mistake.
The method gave the right answer.
- b. The programmer made a mistake.
The method gave the right answer, luckily.
- c. The programmer made a mistake. ✓
The method gave the wrong answer, quietly.
- d. The programmer made a mistake.
The method detected a static error.
- e. The programmer made a mistake.
The method detected a dynamic error.

Your answer is correct.

The correct answer is: The programmer made a mistake.

The method gave the wrong answer, quietly.

Question 6

Correct

Mark 10.00 out of 10.00

We should not use global variables.

Making a variable into a constant can eliminate the risk of global variables.

What keyword should be added to such global variables to make them constants ?

Answer: ✓

The correct answer is: final

Question 7

Incorrect

Mark 0.00 out of 10.00

In the 1990s, the Ariane 5 launch vehicle, designed and built for the European Space Agency, self-destructed 37 seconds after its first launch.

The reason was a control software bug that went undetected. The Ariane 5's guidance software was reused from the Ariane 4, which was a slower rocket. When the velocity calculation converted from a 64-bit floating point number (a `double` in Java terminology, though this software wasn't written in Java) to a 16-bit signed integer (a `short`), it overflowed the small integer and caused an exception to be thrown.

The exception handler had been disabled for efficiency reasons, so the guidance software crashed. Without guidance, the rocket crashed too. The cost of the failure was \$1 billion.

What ideas does this story demonstrate?

Choose the **correct** option.

Select one:

- a. High-quality safety-critical software cannot have residual bugs. ✗
- b. Testing all possible inputs is the best solution to this problem.
- c. Static checking could have detected this bug.
- d. Software exhibits discontinuous behavior, unlike many physically-engineered systems.

Your answer is incorrect.

The correct answer is: Software exhibits discontinuous behavior, unlike many physically-engineered systems.

Question 8

Correct

Mark 10.00 out of 10.00

Consider the following specification:

```
1.  /**
2.   * Reverses the end of a string.
3.   *
4.   *          012345          012345
5.   * For example: reverseEnd("Hello, world", 5) returns "Hello!dlrow ,"
6.   *             <----->           <----->
7.   *
8.   * With start == 0, reverses the entire text.
9.   * With start == text.length(), reverses nothing.
10.  *
11.  * @param text    non-null String that will have its end reversed
12.  * @param start   the index at which the remainder of the input is reversed,
13.  *                requires 0 <= start <= text.length()
14.  * @return input text with the substring from start to the end of the string reversed
15.  */
16. public static String reverseEnd(String text, int start)
```

Which of the following is the **best partitions** for the **start** parameter?

Select one:

- a. start = 0, 0 < start < text.length(), start = text.length() ✓
- b. start = 0, start = 5, start = 100
- c. start < 0, start = 0, start > 0
- d. start < text.length(), start = text.length(), start > text.length()

Your answer is correct.

The correct answer is: start = 0, 0 < start < text.length(), start = text.length()

Question 9

Correct

Mark 10.00 out of 10.00

Consider the following specification:

```

1.  /**
2.   * Reverses the end of a string.
3.   *
4.   *          012345           012345
5.   * For example: reverseEnd("Hello, world", 5) returns "Hello!dlrow ,"
6.   *             <----->           <----->
7.   *
8.   * With start == 0, reverses the entire text.
9.   * With start == text.length(), reverses nothing.
10.  *
11.  * @param text  non-null String that will have its end reversed
12.  * @param start  the index at which the remainder of the input is reversed,
13.  *               requires 0 <= start <= text.length()
14.  * @return input text with the substring from start to the end of the string reversed
15.  */
16. public static String reverseEnd(String text, int start)

```

Which of the following is the **best partitions** for the **text** parameter?

Select one:

- a. `text.length() = 0; text.length()-start is odd; text.length()-start is even` ✓
- b. `text contains some letters; text contains no letters, but some numbers; text contains neither letters nor numbers`
- c. `text.length() < 0; text.length() = 0; text.length() > 0`
- d. `text is every possible string from length 0 to 100`

Your answer is correct.

The correct answer is: `text.length() = 0; text.length()-start is odd; text.length()-start is even`

Question 10

Correct

Mark 10.00 out of 10.00

Select the **incorrect** statement about Covering the Partitions:

Select one:

- a. For the `BigInteger multiply` example, using cover each part approach, we can choose 5 test cases. ✓
- b. The full cartesian approach may not be the best because it could produce too many and redundant test cases.
- c. The cover each part approach may not be the best because the function may behave differently for a certain combination of inputs.
- d. For the `max` example, using full Cartesian approach, we can choose less than 75 test cases because not all combinations are possible.

Your answer is correct.

The correct answer is: For the `BigInteger multiply` example, using cover each part approach, we can choose 5 test cases.

Question 11

Correct

Mark 10.00 out of 10.00

In designing the test suite for the Recursive Reverse String problem,
we include the empty string as a test case.

Which testing principle do we use?

Select one:

- a. Choose the boundaries in the partition. ✓
- b. Divide the input space into subdomains.
- c. Choose one test case from each subdomain.
- d. Choose one test case from every legal combination of the partition.

Your answer is correct.

The correct answer is: Choose the boundaries in the partition.

Question 12

Correct

Mark 10.00 out of 10.00

When you write the recursive step of your recursive method,
which part of your code that must be reached by it ?

Answer: base case



The correct answer is: the base case

Question 13

Correct

Mark 10.00 out of 10.00

In solving a problem recursively, you can define a/an

helper method

- ✓ that uses an arbitrary number of parameters.

[◀ CPT204 2223 Lab 4 Video Recordings](#)

Jump to...

[Lab Exercise #4.1 Check Substring ►](#)

Started on Thursday, 16 March 2023, 20:10**State** Finished**Completed on** Thursday, 16 March 2023, 20:19**Time taken** 8 mins 51 secs**Grade** 96.67 out of 110.00 (87.88%)**Question 1**

Correct

Mark 10.00 out of 10.00

Using your favorite code coverage tool, you add test cases one-by-one, until all reachable statements in your code have been executed at least once.

Which coverage guarantee your code has now?

Select one:

- a. Statement coverage ✓
- b. Branch coverage
- c. Path coverage
- d. Unit coverage

Your answer is correct.

The correct answer is: Statement coverage

Question 2

Correct

Mark 10.00 out of 10.00

Consider the following method:

```
1.  /**
2.   * Sort a list of integers in nondecreasing order. Modifies the list so that
3.   * values.get(i) <= values.get(i+1) for all 0<=i<values.length()-1
4.   */
5.  public static void sort(List<Integer> values) {
6.      // choose a good algorithm for the size of the list
7.      if (values.length() < 10) {
8.          radixSort(values);
9.      } else if (values.length() < 1000*1000*1000) {
10.         quickSort(values);
11.     } else {
12.         mergeSort(values);
13.     }
14. }
```

Which test case of the following test cases are likely to be a **boundary value** produced by **white box testing**?

Select one:

- a. [9, 8, 7, 6, 5, 4, 3, 2, 1, 0] ✓
- b. the empty list
- c. [0, 0, 1, 0, 0, 0, 0]
- d. [1, 2, 3]

Your answer is correct.

The correct answer is: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Question 3

Correct

Mark 10.00 out of 10.00

After fixing a bug that caused test case **X** fail,

you need to rerun all your JUnit tests, not just test case **X**.

Select one:

- True ✓
- False

The correct answer is 'True'.

Question 4

Correct

Mark 10.00 out of 10.00

Which one of these testing activities follows the principle of **regression testing**?

Select one:

- a. Changes should be tested against all inputs that induced bugs in earlier versions of the code ✓
- b. Every component in your code should have an associated set of tests that exercises all the corner cases in its specification
- c. Tests should be written before you write the code as a way of checking your understanding of the specification
- d. When a new test exposes a bug, you should run it on all previous versions of the code until you find the version where the bug was introduced

Your answer is correct.

The correct answer is: Changes should be tested against all inputs that induced bugs in earlier versions of the code

Question 5

Partially correct

Mark 6.67 out of 20.00

Which of these techniques are useful for choosing test cases in test-first programming, **before** any code is written?

Select one or more:

- Partitioning
- Boundaries
- Black box ✓
- Regression
- Coverage
- White box
- Integration

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: Partitioning, Boundaries, Black box

Question 6

Correct

Mark 10.00 out of 10.00

Choose the **correct** statement about a regression test case.

Select one:

- a. A regression test case comes from the discovery of a bug ✓
- b. A regression test case is chosen from the partitions
- c. A regression test case can come out of black-box testing
- d. A regression test case can come out of white-box testing

Your answer is correct.

The correct answer is: A regression test case comes from the discovery of a bug

Question 7

Correct

Mark 10.00 out of 10.00

As a temporary substitute for a method that is not yet to be developed, you write a code to simulate the method's functionality. The method can then be called by another method that you want to test.

Such method is called a/an

Stub

✓ .

Question 8

Correct

Mark 10.00 out of 10.00

Which button to click to get the Java Visualizer run the next line of your code and show the subsequent visualization?

Select one:

- a. Step Into ✓
- b. Step Over
- c. Step Out
- d. Step Off
- e. Step On

Your answer is correct.

The correct answer is: Step Into

Question 9

Correct

Mark 10.00 out of 10.00

Write one line of Java code that *declares* a MyList pointer named **p** and *initializes* it to the current MyList object.

Do not forget to end it with a semicolon.

Answer: ✓

The correct answer is: MyList p = this;

Question 10

Correct

Mark 10.00 out of 10.00

What is the `println` result of :

1. `MyList3 list = new MyList3(100, null);`
2. `list = new MyList3(200, list);`
3. `list = new MyList3(300, list);`
4. `System.out.println(list.get(0));`

Answer: ✓

The correct answer is: 300

[◀ CPT204 2223 Lab 5 Video Recordings](#)

[Lab Exercise #5.1 MyList Iterative Square Mutate ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 6 : 20-24 March — Specification, Linked List, Generics, Linked-based Deque](#) / [Lecture Quiz #6](#)

Started on Friday, 24 March 2023, 16:02

State Finished

Completed on Friday, 24 March 2023, 16:16

Time taken 14 mins 38 secs

Grade **93.33** out of 110.00 (**84.85%**)

Question **1**

Correct

Mark 10.00 out of 10.00

Consider the two methods to find the value **val** in an integer array **a** below.

```
1. static int findFirst(int[] a, int val) {  
2.     for (int i = 0; i < a.length; i++) {  
3.         if (a[i] == val) return i;  
4.     }  
5.     return a.length;  
6. }
```

```
1. static int findLast(int[] a, int val) {  
2.     for (int i = a.length -1 ; i >= 0; i--) {  
3.         if (a[i] == val) return i;  
4.     }  
5.     return -1;  
6. }
```

If clients only care about calling the **find** method when they know that **val** always occurs exactly once in **a**, **findFirst** and **findLast** are behaviorally equivalent.

Select one:

True ✓

False

The correct answer is 'True'.

Question 2

Correct

Mark 10.00 out of 10.00

Consider the two methods to find the value `val` in an integer array `a` below.

```
1. static int findFirst(int[] a, int val) {
2.     for (int i = 0; i < a.length; i++) {
3.         if (a[i] == val) return i;
4.     }
5.     return a.length;
6. }
```

```
1. static int findLast(int[] a, int val) {
2.     for (int i = a.length - 1; i >= 0; i--) {
3.         if (a[i] == val) return i;
4.     }
5.     return -1;
6. }
```

If clients only care that the find method should return any index `i` such that `a[i] == val`, if `val` is in `a`; and any integer `j` where `j` is **not** a valid index of array `a`, otherwise; then `findFirst` and `findLast` are behaviorally equivalent.

Select one:

- True ✓
 False

The correct answer is 'True'.

Question 3

Correct

Mark 10.00 out of 10.00

Suppose we're working on the method below:

```
1. /**
2.  * Requires: tiles has length 7 & contains only uppercase letters.
3.  *           crossings contains only uppercase letters, without duplicates.
4.  * Effects: Returns a list of words where each word can be made by taking
5.  *           letters from tiles and at most 1 letter from crossings.
6.  */
7. public static List<String> scrabble(String tiles, String crossings) {
8.     if (tiles.length() != 7) { throw new RuntimeException(); }
9.     return new ArrayList<>();
10. }
```

Which one is a part of the *postcondition* of `scrabble`?

Select one:

- a. scrabble returns a list of strings✓
 b. tiles has only uppercase letters
 c. crossings has no duplicates
 d. scrabble takes two arguments

Your answer is correct.

The correct answer is: scrabble returns a list of strings

Question 4

Correct

Mark 10.00 out of 10.00

Suppose we're working on the method below:

```

1.  /**
2.   * Requires: tiles has length 7 & contains only uppercase letters.
3.   *           crossings contains only uppercase letters, without duplicates.
4.   * Effects: Returns a list of words where each word can be made by taking
5.   *           letters from tiles and at most 1 letter from crossings.
6.   */
7.  public static List<String> scrabble(String tiles, String crossings) {
8.      if (tiles.length() != 7) { throw new RuntimeException(); }
9.      return new ArrayList<>();
10. }
```

Which one is **not** a part of the *precondition* of **scrabble**?

Select one:

- a. scrabble returns an empty ArrayList✓
- b. tiles has length 7
- c. crossings is a string of uppercase letters
- d. scrabble's arguments are of type String and String

Your answer is correct.

The correct answer is: scrabble returns an empty ArrayList

Question 5

Correct

Mark 10.00 out of 10.00

Suppose we're working on the method below:

```

1.  /**
2.   * Requires: tiles has length 7 & contains only uppercase letters.
3.   *           crossings contains only uppercase letters, without duplicates.
4.   * Effects: Returns a list of words where each word can be made by taking
5.   *           letters from tiles and at most 1 letter from crossings.
6.   */
7.  public static List<String> scrabble(String tiles, String crossings) {
8.      if (tiles.length() != 7) { throw new RuntimeException(); }
9.      return new ArrayList<>();
10. }
```

Which one is the part of the spec that are **checked statically** by Java?

Select one:

- a. scrabble takes two arguments✓
- b. tiles is a string of uppercase letters
- c. crossings has no duplicates
- d. when tiles.length() != 7, scrabble throws a RuntimeException

Your answer is correct.

The correct answer is: scrabble takes two arguments

Question 6

Incorrect

Mark 0.00 out of 10.00

Which of the following is **not** part of a method's specification?

Select one:

- a. restrictions on used data types
- b. return type
- c. restrictions on return values X
- d. number of arguments
- e. argument types
- f. restrictions on argument values

Your answer is incorrect.

The correct answer is: restrictions on used data types

Question 7

Correct

Mark 10.00 out of 10.00

Alice writes the following code:

```

1. public static int gcd(int a, int b) {
2.     if (a > b) {
3.         return gcd(a-b, b);
4.     } else if (b > a) {
5.         return gcd(a, b-a);
6.     }
7.     return a;
8. }
```

Bob writes the following test:

```

1. @Test public void gcdTest() {
2.     assertEquals(6, gcd(24, 54));
3. }
```

Which of the following statement is **incorrect**?

Select one:

- a. If Alice adds $a > 0$ to the precondition, Bob should test negative values of a ✓
- b. If Alice does not add $a > 0$ to the precondition, Bob should test negative values of a
- c. Alice should write $a > 0, b > 0$ in the precondition of gcd
- d. Alice should not write a and b are integers in the precondition of gcd

Your answer is correct.

The correct answer is: If Alice adds $a > 0$ to the precondition, Bob should test negative values of a

Question 8

Partially correct

Mark 13.33 out of 20.00

Given the following specification :

1. static int **find**(int[] arr, int val)
2. requires: arr[0] == val
3. effects: returns index i such that arr[i] == val

Which are the valid test cases for **find** ?

Select one or more:

- find([1, 2, 3], 1) must return 0** ✓
- find([4, 4, 5], 4) must return 0**
- find([4, 4, 5], 4) must return 1**
- find([6, 7, 8], 2) throws an exception** ✗
- find([3], 3) must return 0** ✓
- find([4], 5) must not return 0**

Your answer is partially correct.

You have selected too many options.

The correct answers are: **find([1, 2, 3], 1) must return 0, find([3], 3) must return 0****Question 9**

Correct

Mark 10.00 out of 10.00

What is a condition that must be preserved and guaranteed to be true during a method's execution called ?

Answer: invariant



The correct answer is: invariant

Question 10

Correct

Mark 10.00 out of 10.00

To allow types such as Integer, String, and user-defined types to be a parameter to methods, classes, and interfaces, we use

generics



Using it, we can create classes that work with different data types.

[◀ CPT204 2223 Lab 6 Video Recording](#)

Jump to...

[Lab Exercise #6.1 LLDeque EMPTY CONSTRUCTOR ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 7 : 27-31 March — Exception, Deep Copy, Copy Constructor, Regex, Enum](#) / [Lecture Quiz #7](#)

Started on Friday, 31 March 2023, 14:58

State Finished

Completed on Friday, 31 March 2023, 15:15

Time taken 17 mins 26 secs

Grade 100.00 out of 130.00 (76.92%)

Question 1

Correct

Mark 20.00 out of 20.00

Which of the following **cannot** be null?

Select one or more:

- char c;✓
- static final String str;
- int[] arr;
- Double d;
- final BackAccount myBankAccount;
- String name;
- double d;✓

Your answer is correct.

The correct answers are: char c;,
double d;

Question 2

Correct

Mark 10.00 out of 10.00

Given the following code :

```
1. public static String nope() {  
2.     return null;           // (1)  
3. }  
4.  
5. public static void main(String[] args) {  
6.     String a = nope();      // (2)  
7.     String b = null;        // (3)  
8.     if (a.length() > 0) {   // (4)  
9.         b = a;            // (5)  
10.    }  
11.    return b;             // (6)  
12. }
```

Which line contains a static error ?

Select one:

- (1)
- (2)
- (3)
- (4)
- (5)
- (6) ✓

Your answer is correct.

The correct answer is: (6)

Question 3

Correct

Mark 10.00 out of 10.00

Given the same code from Question 2 above :

```
1. public static String nope() {
2.     return null;           // (1)
3. }
4.
5. public static void main(String[] args) {
6.     String a = nope();    // (2)
7.     String b = null;      // (3)
8.     if (a.length() > 0) { // (4)
9.         b = a;            // (5)
10.    }
11.    return b;             // (6)
12. }
```

Suppose you have commented out the line causing the static error in Question 2.

Now, which line contains a dynamic error ?

Select one:

- (1)
- (2)
- (3)
- (4) ✓
- (5)
- (6)

Your answer is correct.

The correct answer is: (4)

Question 4

Correct

Mark 10.00 out of 10.00

Suppose we're building a robot and we want to specify the method

```
public static List<Point> findPath(Point initial, Point goal)
```

which is responsible for path-finding: determining a sequence of **Points** that the robot should move through to navigate from **initial** to **goal**, past any obstacles that might be in the way.

In the postcondition, we say that **findPath** will search for paths only up to a bounded length (set elsewhere), and that **it will throw an exception if it fails to find one**.

Which exception is the best exception and its type to create, according to Lecture 6?

Select one:

- a. a checked PathNotFoundException ✓
- b. an unchecked PathNotFoundException
- c. a checked NoPathException
- d. an unchecked NoPathException

Your answer is correct.

The correct answer is: a checked PathNotFoundException

Question 5

Correct

Mark 10.00 out of 10.00

Suppose we define a checked exception for the method **findPath**.

What will we choose as our superclass?

Select one:

- a. Exception ✓
- b. Throwable
- c. Error
- d. RuntimeException

Your answer is correct.

The correct answer is: Exception

Question 6

Correct

Mark 10.00 out of 10.00

Suppose we define an unchecked exception for the method `findPath`.

What will we choose as our superclass?

Select one:

- a. Exception
- b. Throwable
- c. Error
- d. RuntimeException✓

Your answer is correct.

The correct answer is: RuntimeException

Question 7

Incorrect

Mark 0.00 out of 20.00

Consider this code below for analyzing some `Thing` objects:

```

1. static List<Thing> allTheThings;
2.
3. static void analyzeEverything() {
4.     analyzeThings();
5. }
6.
7. static void analyzeThings() {
8.     try {
9.         for (Thing t : allTheThings) {
10.             analyzeOneThing(t);
11.         }
12.     } catch (AnalysisException ae) {
13.         return;
14.     }
15. }
16.
17. static void analyzeOneThing(Thing t) throws AnalysisException {
18.     // ...
19.     // ... maybe go past the end of a list
20.     // ...
21. }
```

Note that `IndexOutOfBoundsException`, `NullPointerException`, and `OutOfMemoryError` are unchecked exceptions;

and `AnalysisException` is a checked exception.

Which exception could be thrown by a call to `analyzeEverything`?

Select one or more:

- AnalysisException ✗
- IndexOutOfBoundsException✓
- NullPointerException✓
- OutOfMemoryError

Your answer is incorrect.

The correct answers are: `IndexOutOfBoundsException`, `NullPointerException`, `OutOfMemoryError`

Question 8

Correct

Mark 20.00 out of 20.00

If we want to construct a different object with the same values as the input object, we use a/an

 copy constructor that performs a/an deep copy instead of a shallow copy.**Question 9**

Incorrect

Mark 0.00 out of 10.00

Write one line of Java code that throws an **IllegalArgumentException** object with a message "n must not be even" to complete the if statement below :

```
1. if (n % 2 == 0) {  
2.     // your code here  
3.  
4. }
```

Do not forget to end it with a semicolon.

Answer:



The correct answer is: throw new **IllegalArgumentException("n must not be even")**;

Question 10

Correct

Mark 10.00 out of 10.00

When we throw an **IllegalArgumentException** object within a method, that method must advertise it in the method signature.

Select one:

 True False ✓

The correct answer is 'False'.

[◀ CPT204 2223 Lab 7 Video Recording](#)

[Lab Exercise #7.1 Vehicle CONSTRUCTOR ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 9 : 10-14 April — More Specs, Resizing Array, Circular Array, Array-based Deque](#) / [Lecture Quiz #9](#)

Started on Friday, 14 April 2023, 17:52

State Finished

Completed on Friday, 14 April 2023, 18:11

Time taken 18 mins 48 secs

Grade **85.00** out of 110.00 (77.27%)

Question 1

Partially correct

Mark 5.00 out of 10.00

Consider the following implementation:

```
1. static int findFirst(int[] arr, int val) {
2.     for (int i = 0; i < arr.length; i++) {
3.         if (arr[i] == val) return i;
4.     }
5.     return arr.length;
6. }
```

and this specification of `find`:

```
1. static int find(int[] arr, int val)
2.     requires: nothing
3.     effects: returns largest index i such that
4.             arr[i] == val, or -1 if no such i
```

Which inputs demonstrates that `findFirst` does **not** satisfy this spec?

Select one or more:

- [1, 2, 2], 2 ✓
- [1, 2, 3], 2
- [1, 2, 3], 4
- none of all others, `findFirst` does satisfy this spec!

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: [1, 2, 2], 2, [1, 2, 3], 4

Question 2

Correct

Mark 10.00 out of 10.00

Consider the following implementation:

```
1. static int findLast(int[] arr, int val) {  
2.     for (int i = arr.length -1 ; i >= 0; i--) {  
3.         if (arr[i] == val) return i;  
4.     }  
5.     return -1;  
6. }
```

and this specification of `find`:

```
1. static int find(int[] arr, int val)  
2.     requires: nothing  
3.     effects: returns largest index i such that  
4.             arr[i] == val, or -1 if no such i
```

Which input demonstrates that `findLast` does **not** satisfy this spec?

Select one:

- a. [1, 2, 2], 2
- b. [1, 2, 3], 2
- c. [1, 2, 3], 4
- d. none of all others, `findLast` does satisfy this spec! ✓

Your answer is correct.

The correct answer is: none of all others, `findLast` does satisfy this spec!

Question 3

Correct

Mark 10.00 out of 10.00

For each spec below, which one is **not** deterministic (underdetermined) ?

Select one:

 a.

```
1. static int find(int[] arr, int val)
2.     requires: val occurs in arr
3.     effects: returns index i such that arr[i] == val
```

 b.

```
1. static int find(int[] arr, int val)
2.     requires: val occurs exactly once in arr
3.     effects: returns index i such that arr[i] == val
```

 c.

```
1. static int find(int[] arr, int val)
2.     requires: nothing
3.     effects: returns largest index i such that arr[i] == val, or -1 if no such i
```

 d.

```
1. static int find(int[] arr, int val)
2.     requires: val occurs in arr
3.     effects: returns largest index i such that arr[i] == val
```

Your answer is correct.

The correct answer is:

```
1. static int find(int[] arr, int val)
2.     requires: val occurs in arr
3.     effects: returns index i such that arr[i] == val
```

Question 4

Incorrect

Mark 0.00 out of 10.00

Given this specification:

1. static String `join(String delimiter, String[] elements)`
2. effects: append together the strings `in` elements, but at each step,
`if` there are more elements left, insert delimiter

Rewrite the spec so it is declarative, **not** operational.

Select one:

- a. 1. effects: returns elements joined together with copies `of` delimiter, i.e.
2. `elements[0] + delimiter + elements[1] + delimiter + ... + delimiter + elements[length-1]`
- b. 1. effects: returns the result `of` adding all elements to a
2. `new StringJoiner(delimiter)`
- c. 1. effects: returns the result `of` looping through elements and ~~X~~
2. alternately appending an element and the delimiter
- d. 1. effects: returns the result `of` recursive calls on the elements and
2. `while` concatenating the delimiter

Your answer is incorrect.

The correct answer is:

1. effects: returns elements joined together with copies `of` delimiter, i.e.
2. `elements[0] + delimiter + elements[1] + delimiter + ... + delimiter + elements[length-1]`

Question 5

Correct

Mark 10.00 out of 10.00

When a specification is strengthened :

Select one:

- a. fewer implementations satisfy it, and more clients can use it ✓
- b. fewer implementations satisfy it, and fewer clients can use it
- c. more implementations satisfy it, and fewer clients can use it
- d. more implementations satisfy it, and more clients can use it

Your answer is correct.

The correct answer is: fewer implementations satisfy it, and more clients can use it

Question 6

Correct

Mark 10.00 out of 10.00

Which of the following is **false** about a pair of specifications A and B ?

Select one:

- a. A can be stronger than B and have a stronger precondition ✓
- b. A can be stronger than B and have a weaker precondition
- c. A can be stronger than B and have the same precondition
- d. A can be incomparable to B

Your answer is correct.

The correct answer is: A can be stronger than B and have a stronger precondition

Question 7

Correct

Mark 10.00 out of 10.00

Here are the `find` specifications from Lecture 8 :

1. static int find^{ExactlyOne}(int[] a, int val)
 2. requires: val occurs exactly once `in` a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,AnyIndex}(int[] a, int val)
 2. requires: val occurs at least once `in` a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,FirstIndex}(int[] a, int val)
 2. requires: val occurs at least once `in` a
 3. effects: returns lowest index i such that a[i] == val

1. static int find^{CanBeMissing}(int[] a, int val)
 2. requires: nothing
 3. effects: returns index i such that a[i] == val,
 or -1 if no such i

We already know that `findOneOrMore,FirstIndex` is stronger than `findOneOrMore,AnyIndex`, which is stronger than `findExactlyOne`.

Where is `findExactlyOne` on the diagram ?

Select one:

a.



b.



c.



d.



Your answer is correct.



The correct answer is:

Question 8

Correct

Mark 10.00 out of 10.00

Here are the **find** specifications from Lecture 8 :

1. static int find^{ExactlyOne}(int[] a, int val)
 2. requires: val occurs exactly once **in** a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,AnyIndex}(int[] a, int val)
 2. requires: val occurs at least once **in** a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,FirstIndex}(int[] a, int val)
 2. requires: val occurs at least once **in** a
 3. effects: returns lowest index i such that a[i] == val

1. static int find^{CanBeMissing}(int[] a, int val)
 2. requires: nothing
 3. effects: returns index i such that a[i] == val,
 or -1 if no such i

We already know that **find^{OneOrMore,FirstIndex}** is stronger than **find^{OneOrMore,AnyIndex}**, which is stronger than **find^{ExactlyOne}**.

How does **find^{CanBeMissing}** compare to **find^{ExactlyOne}**?

Select one:

- a. **find^{CanBeMissing}** is stronger than **find^{ExactlyOne}** ✓
- b. **find^{CanBeMissing}** is weaker than **find^{ExactlyOne}**
- c. **find^{CanBeMissing}** and **find^{ExactlyOne}** are incomparable
- d. none of the options is correct

Your answer is correct.

The correct answer is: **find^{CanBeMissing}** is stronger than **find^{ExactlyOne}**

Question 9

Incorrect

Mark 0.00 out of 10.00

Here are the `find` specifications from Lecture 8 :

1. static int find^{ExactlyOne}(int[] a, int val)
 2. requires: val occurs exactly once `in` a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,AnyIndex}(int[] a, int val)
 2. requires: val occurs at least once `in` a
 3. effects: returns index i such that a[i] == val

1. static int find^{OneOrMore,FirstIndex}(int[] a, int val)
 2. requires: val occurs at least once `in` a
 3. effects: returns lowest index i such that a[i] == val

1. static int find^{CanBeMissing}(int[] a, int val)
 2. requires: nothing
 3. effects: returns index i such that a[i] == val,
 or -1 if no such i

We already know that `findOneOrMore,FirstIndex` is stronger than `findOneOrMore,AnyIndex`, which is stronger than `findExactlyOne`.

Where is `findCanBeMissing` on the diagram ?

Select one:

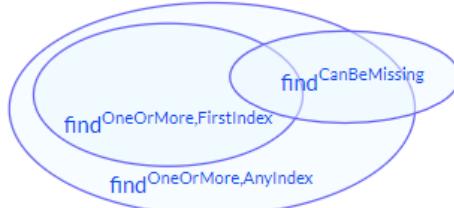
a.



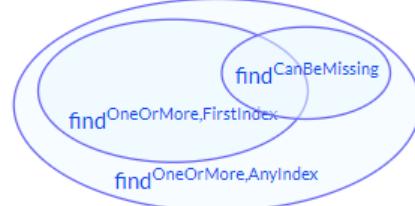
b.

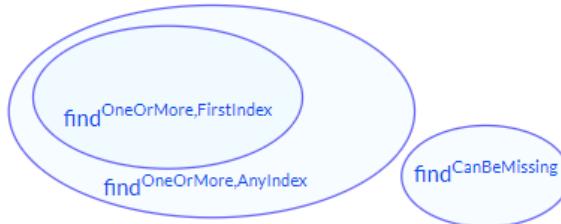


c.

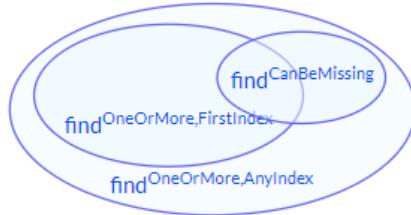


d.



e.

Your answer is incorrect.



The correct answer is:

Question 10

Correct

Mark 10.00 out of 10.00

In our ARList implementation, we use a technique called

resizing array

- ✓ that doubles the size of the array whenever it is full.

Question 11

Correct

Mark 10.00 out of 10.00

You want to use a generic array using casting in your implementation of a data structure.

For example, you write the following line in your constructor or your method:

1. T[] elements = (T[]) new Object[numOfElements];

Write the annotation that you need to write before the constructor or the method:

Answer: @SuppressWarnings("unchecked")



The correct answer is: @SuppressWarnings("unchecked")

[◀ CPT204 2223 Lab 9 Video Recording](#)

Jump to...

[Lab Exercise #9.1 ARDeque EMPTY CONSTRUCTOR ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 10 : 17-21 April — Union Find, Immutability, Defensive Programming, Iterable](#) / [Lecture Quiz #10](#)

Started on Tuesday, 25 April 2023, 15:36

State Finished

Completed on Tuesday, 25 April 2023, 16:26

Time taken 49 mins 46 secs

Marks 110.00/150.00

Grade 117.33 out of 160.00 (73.33%)

Question 1

Correct

Mark 10.00 out of 10.00

Consider the following code, executed in order:

```
1. char text0 = 'a';
2. final char text1 = vowel0;
3.
4. String text2 = text1 + "eiou";
5. final String text3 = text2;
6.
7. char[] text4 = new char[] { text0, 'e', 'i', 'o', 'u' };
8. final char[] text5 = text4;
```

Which of the following statements are legal Java,
that is, produce **no** compiler error if placed **after** the code above?

Select one:

- a. text2 = text3;✓
- b. text1 = text0;
- c. text5 = text4;
- d. text3 = text2;

Your answer is correct.

The correct answer is: text2 = text3;

Question 2

Correct

Mark 10.00 out of 10.00

Consider the following code, executed in order:

```

1. char text0 = 'a';
2. final char text1 = vowel0;
3.
4. String text2 = text1 + "eiuo";
5. final String text3 = text2;
6.
7. char[] text4 = new char[] { text0, 'e', 'i', 'o', 'u' };
8. final char[] text5 = text4;

```

Which of the following statements are legal Java,
that is, produce **no** compiler error if placed **after** the code above?

Select one:

- a. text5[0] = 'x';✓
- b. text2[0] = 'x';
- c. text3[0] = 'x';
- d. text0[0] = 'x';

Your answer is correct.

The correct answer is: text5[0] = 'x';

Question 3

Correct

Mark 10.00 out of 10.00

Consider this (incomplete) method:

```

1. /**
2.  * Solves quadratic equation ax^2 + bx + c = 0.
3.  *
4.  * @param a quadratic coefficient, requires a != 0
5.  * @param b linear coefficient
6.  * @param c constant term
7.  * @return a list of the real roots of the equation
8. */
9. public static List<Double> quadraticRoots(final int a, final int b, final int c) {
10.     List<Double> roots = new ArrayList<Double>();
11.     // A
12.     ... // compute roots
13.     // B
14.     return roots;
15. }

```

What assertion would be reasonable to write at position **A** (*before* computing the roots) ?

Select one:

- a. assert a != 0;✓
- b. assert b != 0;
- c. assert c != 0;
- d. assert roots.size() >= 0;
- e. assert roots.size() <= 2;

Your answer is correct.

The correct answer is: assert a != 0;

Question 4

Correct

Mark 10.00 out of 10.00

Consider this (incomplete) method:

```
1.      /**
2.       * Solves quadratic equation ax^2 + bx + c = 0.
3.       *
4.       * @param a quadratic coefficient, requires a != 0
5.       * @param b linear coefficient
6.       * @param c constant term
7.       * @return a list of the real roots of the equation
8.       */
9.      public static List<Double> quadraticRoots(final int a, final int b, final int c) {
10.          List<Double> roots = new ArrayList<Double>();
11.          // A
12.          ... // compute roots
13.          // B
14.          return roots;
15.      }
```

What assertion would be reasonable to write at position **B** (after computing the roots) ?

Select one:

- a. assert a != 0;
- b. assert b != 0;
- c. assert c != 0;
- d. assert roots.size() >= 0;
- e. assert roots.size() <= 2;✓

Your answer is correct.

The correct answer is: assert roots.size() <= 2;

Question 5

Correct

Mark 10.00 out of 10.00

Consider the following code, which is *missing* some variable declarations :

```
1. class Apartment {
2.
3.     Apartment(String newAddress) {
4.         this.address = newAddress;
5.         this.roommates = new HashSet<Person>();
6.     }
7.
8.     String getAddress() {
9.         return address;
10.    }
11.
12.    void addRoommate(Person newRoommate) {
13.        roommates.add(newRoommate);
14.        if (roommates.size() > MAXIMUM_OCCUPANCY) {
15.            roommates.remove(newRoommate);
16.            throw new TooManyPeopleException();
17.        }
18.    }
19.
20.    int getMaximumOccupancy() {
21.        return MAXIMUM_OCCUPANCY;
22.    }
23. }
```

Which one is the best declaration for the `roommates` variable?

Select one:

- a. final Set<Person> roommates;✓
- b. List<Person> roommates;
- c. Set<Person> roommates;
- d. HashSet<Person> roommates;

Your answer is correct.

The correct answer is: `final Set<Person> roommates;`

Question 6

Correct

Mark 10.00 out of 10.00

Consider the following code, which is *missing* some variable declarations :

```
1. class Apartment {  
2.  
3.     Apartment(String newAddress) {  
4.         this.address = newAddress;  
5.         this.roommates = new HashSet<Person>();  
6.     }  
7.  
8.     String getAddress() {  
9.         return address;  
10.    }  
11.  
12.    void addRoommate(Person newRoommate) {  
13.        roommates.add(newRoommate);  
14.        if (roommates.size() > MAXIMUM_OCCUPANCY) {  
15.            roommates.remove(newRoommate);  
16.            throw new TooManyPeopleException();  
17.        }  
18.    }  
19.  
20.    int getMaximumOccupancy() {  
21.        return MAXIMUM_OCCUPANCY;  
22.    }  
23.}
```

Which one is the best declaration for the `MAXIMUM_OCCUPANCY` variable?

Select one:

- a. static final int MAXIMUM_OCCUPANCY = 8; ✓
- b. final int MAXIMUM_OCCUPANCY = 8;
- c. static int MAXIMUM_OCCUPANCY = 8;
- d. int MAXIMUM_OCCUPANCY = 8;
- e. public int MAXIMUM_OCCUPANCY = 8;
- f. public static int MAXIMUM_OCCUPANCY = 8;

Your answer is correct.

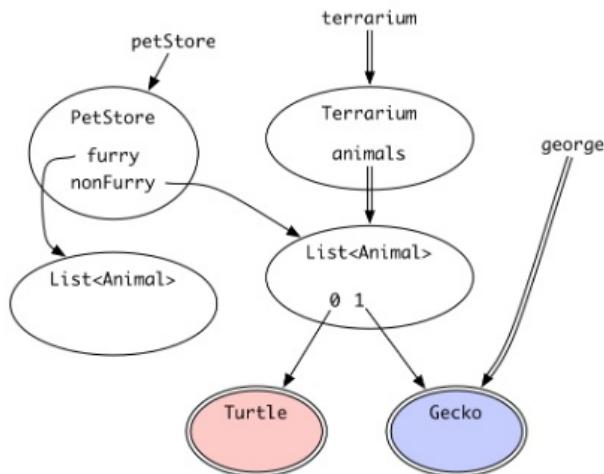
The correct answer is: static final int MAXIMUM_OCCUPANCY = 8;

Question 7

Correct

Mark 10.00 out of 10.00

Consider the following snapshot diagram:



Is it possible that a client with the variable `terrarium` could modify the `Turtle` in red?

Select one:

- a. No, because the "Turtle" is immutable✓
- b. Yes, because all the references between "terrarium" and the "Turtle" are mutable
- c. Yes, because of some reference between "terrarium" and the "Turtle" that is mutable
- d. Yes, because the "Turtle" is mutable
- e. No, because of some reference between "terrarium" and the "Turtle" that is immutable
- f. No, because all the references between "terrarium" and the "Turtle" are immutable

Your answer is correct.

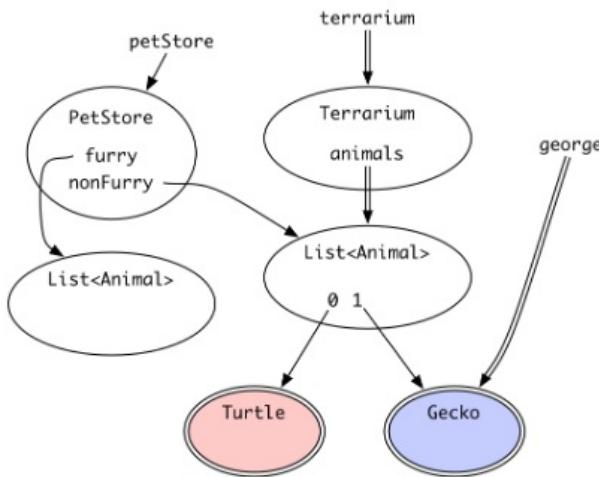
The correct answer is: No, because the "Turtle" is immutable

Question 8

Incorrect

Mark 0.00 out of 10.00

Consider the following snapshot diagram:



Is it possible that a client with the variable `george` could modify the `Gecko` in blue?

Select one:

- a. No, because the "Gecko" is immutable
- b. Yes, because all the references between "george" and the "Gecko" are mutable
- c. Yes, because of some reference between "george" and the "Gecko" that is mutable
- d. Yes, because the "Gecko" is mutable
- e. No, because of some reference between "george" and the "Gecko" that is immutable
- f. No, because all the references between "george" and the "Gecko" are immutable X

Your answer is incorrect.

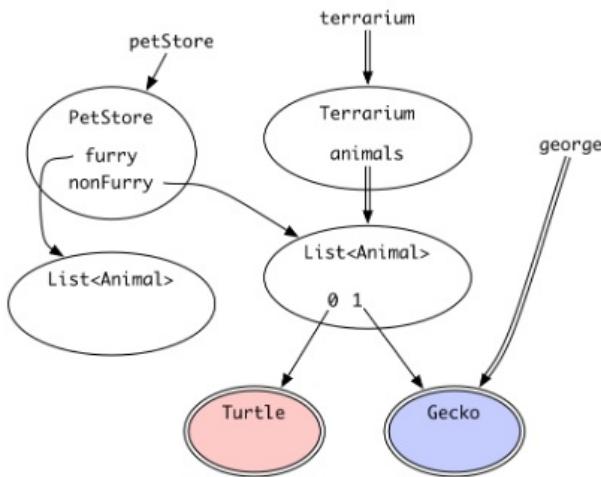
The correct answer is: No, because the "Gecko" is immutable

Question 9

Incorrect

Mark 0.00 out of 10.00

Consider the following snapshot diagram:



Is it possible that a client with the variable `petStore` could do something such that a client with the variable `terrarium` could no longer access the **Gecko** in blue?

Select one:

- a. No, because the "Gecko" is immutable
- b. Yes, because all the references between "petStore" and the "Gecko" are mutable ✗
- c. Yes, because of some reference between "petStore" and the "Gecko" that is mutable
- d. Yes, because the "Gecko" is mutable
- e. No, because of some reference between "petStore" and the "Gecko" that is immutable
- f. No, because all the references between "petStore" and the "Gecko" are immutable

Your answer is incorrect.

The correct answer is: Yes, because of some reference between "petStore" and the "Gecko" that is mutable

Question 10

Incorrect

Mark 0.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        index++;
18.        return element;
19.    }
20. }
```

What is the type of the input to next?

Select one:

- a. Mylterator
- b. void X
- c. ArrayList
- d. String
- e. boolean
- f. int

Your answer is incorrect.

The correct answer is: Mylterator

Question 11

Correct

Mark 10.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        index++;
18.        return element;
19.    }
20. }
```

What is the type of the output to next?

Select one:

- a. MyIterator
- b. void
- c. ArrayList
- d. String ✓
- e. boolean
- f. int

Your answer is correct.

The correct answer is: String

Question 12

Incorrect

Mark 0.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        ++index;
18.        return element;
19.    }
20. }
```

next has the precondition requires: hasNext() returns true.

Which input to next is constrained by the precondition?

Select one:

- a. this
- b. list X
- c. index
- d. element
- e. hasNext

Your answer is incorrect.

The correct answer is: this

Question 13

Correct

Mark 10.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        ++index;
18.        return element;
19.    }
20. }
```

When the precondition is **not** satisfied, the implementation is free to do anything.

What does *this particular implementation* do when the precondition is **not** satisfied?

Select one:

- a. throw an unchecked exception✓
- b. throw a checked exception
- c. return null
- d. return some other element of the list

Your answer is correct.

The correct answer is: throw an unchecked exception

Question 14

Correct

Mark 10.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        ++index;
18.        return element;
19.    }
20. }
```

Part of the postcondition of next is: @return next element of the list.

Which output from next are constrained by that postcondition?

Select one:

- a. the return value ✓
- b. this
- c. hasNext
- d. list

Your answer is correct.

The correct answer is: the return value

Question 15

Correct

Mark 10.00 out of 10.00

Consider MyIterator's next method:

```
1. public class MyIterator {
2.
3.     private final ArrayList<String> list;
4.     private int index;
5.
6.     ...
7.
8.     /**
9.      * Get the next element of the list.
10.     * Requires: hasNext() returns true.
11.     * Modifies: this iterator to advance it to the element
12.     *           following the returned element.
13.     * @return next element of the list
14.     */
15.    public String next() {
16.        final String element = list.get(index);
17.        ++index;
18.        return element;
19.    }
20. }
```

Another part of the postcondition of next is **modifies: this iterator to advance it to the element following the returned element.**

What is constrained by that postcondition?

Select one:

- a. the return value
- b. this✓
- c. hasNext
- d. list

Your answer is correct.

The correct answer is: this

[◀ CPT204 2223 Lab 10 Video Recording](#)

Jump to...

[Lab Exercise #10.1 ARDequilterator CONSTRUCTOR and HASNEXT ►](#)

[Dashboard](#) / My courses / [CPT204-2223-S2](#) / [Sections](#)

/ [Week 11 : 25-29 April — ADT, Interface, Inheritance, Dynamic Method Selection, Set](#) / [Lecture Quiz #11](#)

Started on Thursday, 4 May 2023, 18:44

State Finished

Completed on Thursday, 4 May 2023, 18:51

Time taken 6 mins 51 secs

Grade **70.00** out of 150.00 (**46.67%**)

Question **1**

Incorrect

Mark 0.00 out of 10.00

Consider an abstract data type `Bool`.

The type has the following operations:

1. `true : Bool`
2. `false : Bool`
- 3.
4. `and : Bool × Bool → Bool`
5. `or : Bool × Bool → Bool`
6. `not : Bool → Bool`

where the first two operations construct the two values of the type,

and last three operations have the usual meanings of logical `and`, logical `or`, and logical `not` on those values.

The following are possible ways that `Bool` might be implemented and still be able to satisfy the specs of the operations, except one.

Which one is **not** the correct way?

Select one:

- a. As a `long` value in which all possible values mean `true`.
- b. As a single bit, where 1 means `true` and 0 means `false`.
- c. As an `int` value where 2 means `true` and 5 means `false`.
- d. As a reference to a `String` object where "false" to mean `true` and "true" to mean `false` 

Your answer is incorrect.

The correct answer is: As a `long` value in which all possible values mean `true`.

Question 2

Correct

Mark 10.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

`Integer.valueOf()`

<https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html#valueOf-java.lang.String->

Select one:

- a. creator✓
- b. producer
- c. mutator
- d. observer

Your answer is correct.

The correct answer is: creator

Question 3

Correct

Mark 10.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

`BigInteger.mod()`

<https://docs.oracle.com/javase/8/docs/api/java/math/BigInteger.html#mod-javax.math.BigInteger->

Select one:

- a. creator
- b. producer✓
- c. mutator
- d. observer

Your answer is correct.

The correct answer is: producer

Question 4

Correct

Mark 10.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

List.addAll()

<https://docs.oracle.com/javase/8/docs/api/java/util/List.html#addAll-java.util.Collection->

Select one:

- a. creator
- b. producer
- c. mutator✓
- d. observer

Your answer is correct.

The correct answer is: mutator

Question 5

Incorrect

Mark 0.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

Collections.unmodifiableList()

<https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#unmodifiableList-java.util.List->

Select one:

- a. creator✗
- b. producer
- c. mutator
- d. observer

Your answer is incorrect.

The correct answer is: producer

Question 6

Correct

Mark 10.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

String.toUpperCase()

<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toUpperCase-->

Select one:

- a. creator
- b. producer✓
- c. mutator
- d. observer

Your answer is correct.

The correct answer is: producer

Question 7

Correct

Mark 10.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

Set.contains()

<https://docs.oracle.com/javase/8/docs/api/java/util/Set.html#contains-java.lang.Object->

Select one:

- a. creator
- b. producer
- c. mutator
- d. observer✓

Your answer is correct.

The correct answer is: observer

Question 8

Incorrect

Mark 0.00 out of 10.00

The method below is an operation on an abstract data type from the Java library.

It is followed by the link of its documentation.

Read it, and classify the operation :

BufferedReader.readLine()

<https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html#readLine-->

Select one:

- a. creator
- b. producer X
- c. mutator
- d. observer

Your answer is incorrect.

The correct answer is: mutator

Question 9

Incorrect

Mark 0.00 out of 10.00

Consider the following abstract data type.

```

1.  /**
2.   * Represents a family that lives in a household together.
3.   * A family always has at least one person in it.
4.   * Families are mutable.
5.   */
6.  class Family {
7.      // the people in the family, sorted from oldest to youngest, with no duplicates.
8.      public List<Person> people;
9.
10.     /**
11.      * @return a list containing all the members of the family, with no duplicates.
12.      */
13.      public List<Person> getMembers() {
14.          return people;
15.      }
16.  }
```

Here is a client of this abstract data type:

```

1. void client1(Family f) {
2.     // get youngest person in the family
3.     Person baby = f.people.get(f.people.size() - 1);
4.     ...
5. }
```

Assume all this code works correctly (both `Family` and `client1`) and passes all its tests.

Now `Family`'s representation is changed from a `List` to `Set`, as shown:

```

1.  /**
2.   * Represents a family that lives in a household together.
3.   * A family always has at least one person in it.
4.   * Families are mutable.
5.   */
6.  class Family {
7.      // the people in the family
8.      public Set<Person> people;
9.
10.     /**
11.      * @return a list containing all the members of the family, with no duplicates.
12.      */
13.      public List<Person> getMembers() {
14.          return new ArrayList<>(people);
15.      }
16.  }
```

Assume that `Family` compiles correctly after the change.

Which of the following statements are true about `client1` after `Family` is changed?

Select one:

- a. `client1` is independent of `Family`'s representation, so it keeps working correctly.
- b. `client1` depends on `Family`'s representation, and the dependency would be caught as a static error. ✗
- c. `client1` depends on `Family`'s representation, and the dependency would be caught as a dynamic error.
- d. `client1` depends on `Family`'s representation, and the dependency would not be caught but would produce a wrong answer at runtime.
- e. `client1` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answer.

Your answer is incorrect.

The correct answer is: `client1` depends on `Family`'s representation, and the dependency would be caught as a static error.

Question 10

Incorrect

Mark 0.00 out of 10.00

Consider the following abstract data type.

```

1.  /**
2.   * Represents a family that lives in a household together.
3.   * A family always has at least one person in it.
4.   * Families are mutable.
5.   */
6.  class Family {
7.      // the people in the family, sorted from oldest to youngest, with no duplicates.
8.      public List<Person> people;
9.
10.     /**
11.      * @return a list containing all the members of the family, with no duplicates.
12.      */
13.     public List<Person> getMembers() {
14.         return people;
15.     }
16. }
```

Here is a client of this abstract data type:

```

1.  void client2(Family f) {
2.      // get size of the family
3.      int familySize = f.people.size();
4.      ...
5. }
```

Assume all this code works correctly (both `Family` and `client2`) and passes all its tests.

Now `Family`'s representation is changed from a `List` to `Set`, as shown:

```

1.  /**
2.   * Represents a family that lives in a household together.
3.   * A family always has at least one person in it.
4.   * Families are mutable.
5.   */
6.  class Family {
7.      // the people in the family
8.      public Set<Person> people;
9.
10.     /**
11.      * @return a list containing all the members of the family, with no duplicates.
12.      */
13.     public List<Person> getMembers() {
14.         return new ArrayList<>(people);
15.     }
16. }
```

Assume that `Family` compiles correctly after the change.

Which of the following statements are true about `client2` after `Family` is changed?

Select one:

- a. `client2` is independent of `Family`'s representation, so it keeps working correctly.
- b. `client2` depends on `Family`'s representation, and the dependency would be caught as a static error.
- c. `client2` depends on `Family`'s representation, and the dependency would be caught as a dynamic error.
- d. `client2` depends on `Family`'s representation, and the dependency would not be caught but would produce a wrong answer at runtime. ✗
- e. `client2` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answer.

Your answer is incorrect.

The correct answer is: `client2` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answer.

Question 11

Incorrect

Mark 0.00 out of 10.00

Consider the following abstract data type.

```

1.  /**
2.   * Represents a family that lives in a household together.
3.   * A family always has at least one person in it.
4.   * Families are mutable.
5.   */
6.  class Family {
7.      // the people in the family, sorted from oldest to youngest, with no duplicates.
8.      public List<Person> people;
9.
10.     /**
11.      * @return a list containing all the members of the family, with no duplicates.
12.      */
13.     public List<Person> getMembers() {
14.         return people;
15.     }
16. }
```

Here is a client of this abstract data type:

```

1. void client3(Family f) {
2.     // get any person in the family
3.     Person anybody = f.getMembers().get(0);
4.     ...
5. }
```

Assume all this code works correctly (both `Family` and `client3`) and passes all its tests.

Now `Family`'s representation is changed from a `List` to `Set`, as shown:

```

1. /**
2.  * Represents a family that lives in a household together.
3.  * A family always has at least one person in it.
4.  * Families are mutable.
5.  */
6. class Family {
7.     // the people in the family
8.     public Set<Person> people;
9.
10.    /**
11.       * @return a list containing all the members of the family, with no duplicates.
12.       */
13.     public List<Person> getMembers() {
14.         return new ArrayList<>(people);
15.     }
16. }
```

Assume that `Family` compiles correctly after the change.

Which of the following statements are true about `client3` after `Family` is changed?

Select one:

- a. `client3` is independent of `Family`'s representation, so it keeps working correctly.
- b. `client3` depends on `Family`'s representation, and the dependency would be caught as a static error.
- c. `client3` depends on `Family`'s representation, and the dependency would be caught as a dynamic error.
- d. `client3` depends on `Family`'s representation, and the dependency would not be caught but would produce a wrong answer at runtime. ✗
- e. `client3` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answer.

Your answer is incorrect.

The correct answer is: `client3` is independent of `Family`'s representation, so it keeps working correctly.

Question 12

Correct

Mark 10.00 out of 10.00

```
1. 1 /**
2.  * Represents a family that lives in a household together.
3.  * A family always has at least one person in it.
4.  * Families are mutable.
5. */
6 public class Family {
7     // the people in the family, sorted from oldest to youngest, with no duplicates.
8     private List<Person> people;
9
10    /**
11     * @return a list containing all the members of the family, with no duplicates.
12     */
13    public List<Person> getMembers() {
14        return people;
15    }
16 }
```

Which line is part of the representations?

Select one:

- a. lines 1-5
- b. line 6
- c. line 8 ✓
- d. lines 10-12
- e. line 13
- f. line 14

Your answer is correct.

The correct answer is: line 8

Question 13

Incorrect

Mark 0.00 out of 10.00

```

1. 1 /**
2.  * Represents a family that lives in a household together.
3.  * A family always has at least one person in it.
4.  * Families are mutable.
5. */
6 public class Family {
7     // the people in the family, sorted from oldest to youngest, with no duplicates.
8     private List<Person> people;
9
10    /**
11     * @return a list containing all the members of the family, with no duplicates.
12     */
13    public List<Person> getMembers() {
14        return people;
15    }
16 }
```

Which line is part of the implementations?

Select one:

- a. lines 1-5
- b. line 6
- c. line 8 X
- d. lines 10-12
- e. line 13
- f. line 14

Your answer is incorrect.

The correct answer is: line 14

Question 14

Incorrect

Mark 0.00 out of 10.00

Choose the correct statement.

Select one:

- a. If you are a subclass of an interface, you have to override all of its method signatures.
- b. If you override a method, you have to annotate the method with @Override X
- c. Method overloading is when you have multiple methods with the same signature, but different names.
- d. An object o is instantiated with static type S and dynamic type D.
D is a subclass of S, and D overloads method m() of S.
At runtime, o.m() will call method m() that belongs to D.

Your answer is incorrect.

The correct answer is: If you are a subclass of an interface, you have to override all of its method signatures.

Question 15

Correct

Mark 10.00 out of 10.00

Which statement is **incorrect** about default method ?

Select one:

- a. Default method must be overridden by the subclass of the interface. ✓
- b. Default method is implemented in an interface.
- c. Default method is inherited by the subclass of the interface.
- d. Default method that is overridden by a subclass of the interface will be run because of the dynamic method selection.

Your answer is correct.

The correct answer is: Default method must be overridden by the subclass of the interface.

[◀ CPT204 2223 Lab 11 Video Recording](#)

Jump to...

[Lab Exercise #11.1 ARDeque DEQUE INTERFACE ►](#)

Started on Thursday, 11 May 2023, 14:31

State Finished

Completed on Thursday, 11 May 2023, 14:36

Time taken 4 mins 45 secs

Grade 16.00 out of 30.00 (53.33%)

Question 1

Partially correct

Mark 6.00 out of 10.00

Consider the latest implementation of Duration in the lecture:

```

1. public class Duration {
2.     private final int mins;
3.     private final int secs;
4.
5.     /** Make a duration lasting for m minutes and s seconds. */
6.     public Duration(int m, int s) {
7.         mins = m; secs = s;
8.     }
9.     /** @return length of this duration in seconds */
10.    public long getLength() {
11.        return mins*60 + secs;
12.    }
13.
14.    private static final int CLOCK_SKEW = 5; // seconds
15.
16.    @Override
17.    public boolean equals (Object thatObject) {
18.        if (!(thatObject instanceof Duration)) return false;
19.        Duration thatDuration = (Duration) thatObject;
20.        return Math.abs(this.getLength() - thatDuration.getLength()) <= CLOCK_SKEW;
21.    }
22. }
```

Suppose these Duration objects are created:

```

1. Duration d_0_60 = new Duration(0, 60);
2. Duration d_1_00 = new Duration(1, 0);
3. Duration d_0_57 = new Duration(0, 57);
4. Duration d_1_03 = new Duration(1, 3);
```

Which of the following expressions return true?

Select one or more:

- i. d_0_57.equals(d_1_03)
- ii. d_0_60.equals(d_1_00) ✓
- iii. d_1_00.equals(d_0_60) ✓
- iv. d_1_00.equals(d_1_00) ✓
- v. d_0_57.equals(d_1_00)
- vi. d_0_60.equals(d_1_03)

Your answer is partially correct.

You have correctly selected 3.

The correct answers are: d_0_60.equals(d_1_00), d_1_00.equals(d_0_60), d_1_00.equals(d_1_00), d_0_57.equals(d_1_00), d_0_60.equals(d_1_03)

Question 2

Incorrect

Mark 0.00 out of 10.00

Consider the latest implementation of Duration in the lecture:

```

1. public class Duration {
2.     private final int mins;
3.     private final int secs;
4.
5.     /** Make a duration lasting for m minutes and s seconds. */
6.     public Duration(int m, int s) {
7.         mins = m; secs = s;
8.     }
9.     /** @return length of this duration in seconds */
10.    public long getLength() {
11.        return mins*60 + secs;
12.    }
13.
14.    private static final int CLOCK_SKEW = 5; // seconds
15.
16.    @Override
17.    public boolean equals (Object thatObject) {
18.        if (!(thatObject instanceof Duration)) return false;
19.        Duration thatDuration = (Duration) thatObject;
20.        return Math.abs(this.getLength() - thatDuration.getLength()) <= CLOCK_SKEW;
21.    }
22. }
```

Which properties of an equivalence relation are violated by this `equals()` method?

Select one:

- a. recursivity
- b. reflexivity ✖
- c. sensitivity
- d. symmetry
- e. transitivity

Your answer is incorrect.

The correct answer is: transitivity

Question 3

Correct

Mark 10.00 out of 10.00

Suppose you want to show that an equality operation is buggy because it is **not** reflexive.

How many objects do you need for a counterexample to reflexivity?

Select one:

- a. 0 objects
- b. 1 object ✓
- c. 2 objects
- d. 3 objects
- e. 4 objects

Your answer is correct.

The correct answer is: 1 object

[◀ CPT204 2223 Lab 12 Video Recording](#)

Jump to...

[Lab Exercise #12.1 Duration COMPARETO ►](#)

Started on Thursday, 18 May 2023, 23:41

State Finished

Completed on Thursday, 18 May 2023, 23:43

Time taken 2 mins 2 secs

Grade 10.00 out of 30.00 (33.33%)

Question 1

Correct

Mark 10.00 out of 10.00

Here is the code again from the slide Autoboxing and Equality:

```
1. Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();
2. a.put("c", 130); // put ints into the map
3. b.put("c", 130);
```

What is the compile-time type of the expression 130?

After executing `a.put("c", 130)`, what is the runtime type that is used to represent the value 130 in the map?

What is the compile-time type of `a.get("c")`?

Select one:

- a. int, Integer, Integer✓
- b. int, Integer, int
- c. Integer, int, int
- d. int, int, Integer
- e. Integer, int, Integer
- f. Integer, Integer, int

Your answer is correct.

30 is an integer literal, so its compile-time type is `int`.

In the `Map<String, Integer>`, the keys are `Strings` and the values are `Integers`. So when 130 is placed in the map, it is automatically boxed up into a fresh `Integer` object.

The `get()` operation for a `Map<K, V>` returns values of type `V`, so for a `Map<String, Integer>`, the type would be `Integer`.

The correct answer is: `int, Integer, Integer`

Question 2

Incorrect

Mark 0.00 out of 10.00

Here is the code again from the slide Autoboxing and Equality:

```
1. Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();
2. a.put("c", 130); // put ints into the map
3. b.put("c", 130);
```

After this code executes, what would `a.get("c").equals(b.get("c"))` return?

What would `a.get("c") == b.get("c")` return?

Select one:

- a. true, false
- b. false, true
- c. true, true X
- d. false, false

Your answer is incorrect.

Both `get()` calls return an `Integer` object representing 130. Since `equals()` is correctly implemented for the (immutable) `Integer` type, it returns `true` for those two values.

The `get()` calls return *distinct* `Integer` objects, so they are not referentially equal. `==` returns `false`.

This is the surprising pitfall: if you have in your mind that the `Map` contains `int` values, you will be surprised by the behavior of `get()`, because it returns an `Integer` instead. Most of the time you can use `Integer` interchangeably with `int`, but not when it comes to equality operators like `==` and `equals`.

The correct answer is: true, false

Question 3

Incorrect

Mark 0.00 out of 10.00

Here is the code again from the slide Autoboxing and Equality:

```
1. Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();
2. a.put("c", 130); // put ints into the map
3. b.put("c", 130);
```

Now suppose you assign the `get()` results to `int` variables:

```
1. int i = a.get("c");
2. int j = b.get("c");
3. boolean isEqual = (i == j);
```

Is there an error with that code, or if not, what is the value of `isEqual`?

Select one:

- a. true
- b. false
- c. compile error ✘
- d. runtime error

Your answer is incorrect.

The assignments automatically *unbox* the `Integer` objects into `int` values, both 130. Those primitive `int` values are both 130, so `==` now returns `true`.

Behavior differences like this make autoboxing/unboxing bugs hard to spot and easy to introduce. Another reason they can be tricky: if we asked these same questions with `127` instead of `130`, the answers would be different! For the integers from -128 to 127, the boxed `Integer` objects come from a pool that is reused every time, and the objects will be `==`.

The correct answer is: true

[◀ CPT204 2223 Lab 13 Video Recording](#)

Jump to...

[Lab Exercise #13.1 HASet SIZECOMPARATOR ►](#)

CPT204 Review

Quiz 错题

1. Quiz5Q3 : After fixing a bug that caused test case fail, you need to rerun all your JUnit tests, not just test case X.(True)
2. Changes should be tested against all inputs that induced bugs in earlier versions of the code follows ***regression testing***.
3. in test-first programming, before any code is written, ***Partitioning, Boundaries, Black box*** are very useful techniques.
4. ***A regression test case comes from the discovery of a bug.***
5. As a temporary substitute for a method that is not yet to be developed, you write a code to simulate the method's functionality. The method can then be called by another method that you want to test. Such method is called a/an ***Stub***.
6. Which button to click to get the Java Visualizer run the next line of your code and show the subsequent visualization? ----***Step Into***
7. Part of a method's specification:
 - a) restrictions on used data types (***isn't one part***)
The below are parts in specification.
 - b) return type.
 - c) Restrictions on return values.
 - d) Numbers of arguments.
 - e) Argument types.
 - f) Restrictions on argument values.
8. ***Invariant*** is a condition that must be preserved and guaranteed to be true during a method's execution.
9. To allow types such as Integer, String, and user-defined types to be a parameter to methods, classes, and interfaces, we use ***generics***. Using it, we can create classes that work with different data types.
- 10.

1. OpenAI

Token:

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpxVCIsImtpZCI6Ik1UaEVOVUpHTkVNMVFU
RTRNMEZCTWpkQ05UZzVNRFUxUlRVd1FVSkRNRU13UmtGRVFrRXpSZyJ9.e
yJodHRwczovL2FwaS5vcGVuYWkuY29tL3Byb2ZpbGUiOnsiZW1haWwiOiJuaW1
pbmVAb3V0bG9vay5jb20iLCJlbWFpbF92ZXJpZmllZCI6dHJ1ZX0sImh0dHBzOi8v
YXBpLm9wZW5haS5jb20vYXV0aCI6eyJ1c2VyX2lkIjoidXNlc1ZQVJySVVVU01
udlEwZDBYeFU1b016V2UiSwiaXNzljoiaHR0cHM6Ly9hdXR0MC5vcGVuYWku
Y29tLyIsInN1YiI6ImF1dGgwDYzOGYZY2IzOTdlNjZhMTY2MjlhYTFhMSIsImF
1ZCI6WyJodHRwczovL2FwaS5vcGVuYWkuY29tL3YxIiwiaHR0cHM6Ly9vcGVu
YWkub3BlbmFpLmF1dGgwYXBwLmNvbS91c2VyaW5mbvJdLCJpYXQiOjE2OD
M3MTczNDksImV4cCI6MTY4NDkyNjk0OSwiYXpwIjoiVGRKSWNiZTE2V29US

HROOTVueXl3aDVFNHlPbzZJdEciLCJzY29wZSI6Im9wZW5pZCBwcm9maWxII
GVtYWlsIG1vZGVsLnJIYWQgbW9kZWwucmVxdWVzdCBvcmdhbml6YXRpb24
ucmVhZCBvZmZsaW5IX2FjY2VzcyJ9.J6_eXRHKCVtOfzFYdqGoEnCF5Zo0KYD
ZVgH8MpYUNQHVpD_Cb_ourBjIFNv3S8kXG7kh3FR_LSz2ij1zKusMuMieMrV
E7rMC2oY-
Mrs7Xah1NFykqkdRzuraZvRK1zpboMMcMJZDuUTm8b9Xum9O_Re5XC7I9n_W
0QCC-zSgSOVRPIC6-edgTZxNCUl0g1Cq69Xz8nEFqZXSBd8-
HHWsa_OfpvuGCPjTtjCqrMWQP4DkIZdgNmk5Zd7JYacCdBr6hW94Me1EiV4V4
zS_3zVrIOVmbmVK-VV8hkmVpv8Fk-
IYB4QkdSSiAPBerAFHnr3uAF7p2iRGMYVSDLAbEpRovw

3zVrIOVmbmVK-VV8hkmVpv8Fk-
IYB4QkdSSiAPBerAFHnr3uAF7p2iRGMYVSDLAbEpRovw
API: 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

2. Lecture 2

1. Java是一种statically-typed的语言
2. Static Checking:
 - a) Static typing是一种特殊的**static checking**方式，会在**compile time** check bug。
3. String的相关用法复习
4. Array的相关用法复习:
 - a) index: a [2]
 - b) assignment: a [2] = 0
 - c) length: a.length
5. List type:
 - a) List是一个其他type的sequence
 - b) List的声明: List<Integer> list = new ArrayList<Integer>();
 - c) List的一些用法:
 - i. Index: list.get(2)
 - ii. assignment list.set(2, 5)
 - iii. add: list.add(5)
 - iv. length list.size()

3. Lecture 3

1. Checking 的形式主要有三种：
 - a) Static checking: 在程序运行前就会被发现
 - b) Dynamic checking: 当代码执行后才会被发现
 - c) No checking: Java 语言不会发现这种错误，但是结果是错误的
2. Static checking:
 - a) syntax errors
 - b) wrong names, e.g., Math.sine(2), The right name is sin
 - c) wrong number of arguments, e.g. Math. Sin(30, 20)
3. Dynamic checking
 - a) illegal argument values 例如 x/y 但 $y=0$ 但如果是 double 0 在分母就是 no checking
 - b) unrepresentable return values
 - c) out of range indexes
 - d) calling a method on a null object reference
4. No checking
 - a) Integer division: 例如 $5/2$ 不会返回小数部分，会返回整数部分 2
 - b) Integer overflow: 超出了 int, long 这些类型的范围
 - c) Special values in floating point types.
5. Javadoc comments
6. Two goals:
 - a) Communicating with the computer
 - b) Communicating with other people
7. Quiz 错题：
 - a) Double,double 可能有 no checking 的情况
 - b) object pointed by final variable **can** be mutated

4. Lecture 4

1. Coding rules
 - a) Don't repeat yourself
 - b) Comments where needed
 - c) Fail fast
 - d) Avoid magic numbers: 不要用数字做变量名
 - e) One purpose for each variable
 - f) Use good names
 - g) Don't use global variables
 - h) Return results, don't print them
 - i) Use white space for 可读性
2. Systematic testing
 - a) Systematic testing 是用 test case 来测试完整的系统“
 - b) 选择的 case 有如下要求：
 - i. Correct
 - ii. Thorough
 - iii. Small

5. Lecture 5

1. Testing
2. Coverage 覆盖率
 - a) 查看测试覆盖了多少代码
 - b) 有三种常用的 coverage
 - i. Path > Branch > Statement
3. Unit testing: 测试每一个单独的 module
4. Integration testing: 测试整体
5. Automated regression testing

6. Lecture 6

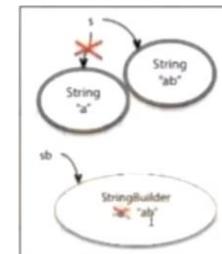
1. Specification: 对 Javadoc 写法的约束

7.Lecture 7

8.Lecture 10

1. Make bugs impossible
2.
 - a) Immutable type (无法被改变的值): 例如 **String**
 - b) Immutable reference: **final**
 - c) **Reference** 是 immutable 不代表里面的值不可以被 change
3. Assertions (断言) (做了解)
 - a) Runtime assert statement: eg. assert ($x \geq 0$) ...
 - b) Assertion 是 off by default 的, 因为它会消耗大量资源
 - c) Assertion 一般写在 test 中
4. 模块化系统设计: **Modularity**
5. 非模块化的设计: **Monolithic**
6. Encapsulation (封装)
 - a) Access control
 - b) Variable scope (定义一些局部变量) 例如 在 for-loop 中尽量把 int i 定义在括号中而不是在外面声明

- c) 最小化 Variable scope 的一个方法就是少用 global variable
- 7. String 是一个 immutable 的变量，而 StringBuilder 是 mutable 的。因此 String 中的内容修改后被调用会指向一个新的内存地址，而 StringBuilder 不会。会指向同一个内存地址
- 8. A same mutable object with multiple reference is called **aliases**.



9. Iterator 迭代器

- a) 迭代器有两个重要方法：
 - i. `Next()`
 - ii. `hasNext()`

9.Lecture 11

- 1. ADT 和 Rep Independence 的区别
- 2. 每个抽象类一般有 4 个主要方法