Started on	Wednesday, 10 March 2021, 19:49
State	Finished
Completed on	Sunday, 14 March 2021, 17:14
Time taken	3 days 21 hours
Grade	**80.00** out of 150.00 (**53**%)

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?

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

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

need boolean, but is int

Question 2

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?

```
int bigNum = 200000;        // bigNum is 200,000
bigNum = bigNum * bigNum;   // bigNum should be 4 billion
```

Select one:

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

result: 1345294336

Your answer is incorrect.

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

Question 3

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?

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

Select one:

a. static checking

b. dynamic checking

c. no checking, resulting in wrong answer

0.0
integer division in floating-point context

Your answer is incorrect.

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?

```
int sum = 0;
int n = 0;
int average = sum/n;
```

Select one:

a. static checking

b. dynamic checking

c. no checking, resulting in wrong answer

java.lang.ArithmeticException: / by zero

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?

```
double sum = 7;
double n = 0;
double average = sum/n;
```

Select one:

a. static checking

b. dynamic checking

c. no checking, resulting in wrong answer Infinity

Your answer is incorrect.

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

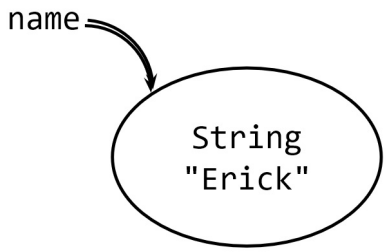
Question 6

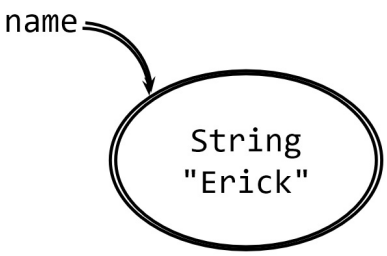
Correct Mark 10.00 out of 10.00

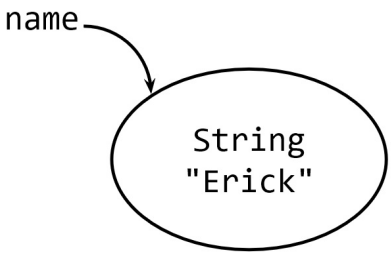
Which is the correct snapshot diagram for:

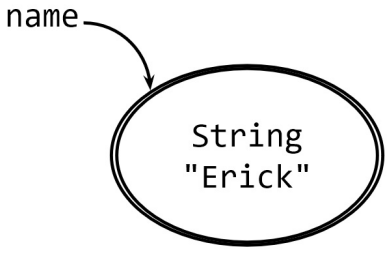
final String name = "Erick";

Select one:









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

Incorrect Mark 0.00 out of 20.00

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

Therefore, final provides you immutable reference ✗ for immutable values ✗ .

static checking references

Question 9

Incorrect Mark 0.00 out of 10.00

Rewrite the variable declaration below using Lists instead of arrays:

```
char[][] matrix;
```

Answer: List matrix = new ArrayList<>() ✗

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

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(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 correct.

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");

Finish review

Lab 2 Videos Jump to... Lab Exercise 2.1 Max Stret

Started on	Monday, 22 March 2021, 19:14
State	Finished
Completed on	Monday, 22 March 2021, 19:41
Time taken	27 mins
Grade	**90.00** out of 130.00 (**69**%)

Question 1

Correct Mark 10.00 out of 10.00

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

```
public static int dayOfYear(int month, int dayOfMonth, int year) {
    if (month == 2) {
        dayOfMonth += 31;
    } else if (month == 3) {
        dayOfMonth += 59;
    } else if (month == 4) {
        dayOfMonth += 90;
    } else if (month == 5) {
        dayOfMonth += 31 + 28 + 31 + 30;
    } else if (month == 6) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31;
    } else if (month == 7) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30;
    } else if (month == 8) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31;
    } else if (month == 9) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31;
    } else if (month == 10) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30;
    } else if (month == 11) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31;
    } else if (month == 12) {
        dayOfMonth += 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 31;
    }
    return dayOfMonth;
}
```

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:

```
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

Incorrect | Mark 0.00 out of 10.00

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

```
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 incorrect.
The correct answer is: The programmer made a mistake.
The method gave the right answer, luckily.

Question 3

Incorrect | Mark 0.00 out of 10.00

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

```
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 incorrect.

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:

```
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:

```
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

Incorrect Mark 0.00 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: local ✗

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 slo 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. 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:

```
/**
 * Reverses the end of a string.
 *
 *                          012345                    012345
 * For example: reverseEnd("Hello, world", 5) returns "Hellodlrow ,"
 *                               <-----                    <-----
 *
 * With start == 0, reverses the entire text.
 * With start == text.length(), reverses nothing.
 *
 * @param text    non-null String that will have its end reversed
 * @param start   the index at which the remainder of the input is reversed,
 *                requires 0 <= start <= text.length()
 * @return input text with the substring from start to the end of the string reversed
 */
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:

```
/**
 * Reverses the end of a string.
 *
 *                          012345                    012345
 * For example: reverseEnd("Hello, world", 5) returns "Hellodlrow ,"
 *                               <-----                    <-----
 *
 * With start == 0, reverses the entire text.
 * With start == text.length(), reverses nothing.
 *
 * @param text    non-null String that will have its end reversed
 * @param start   the index at which the remainder of the input is reversed,
 *                requires 0 <= start <= text.length()
 * @return input text with the substring from start to the end of the string reversed
 */
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.

Finish review

Lab 3 Recording Jump to... Lab Exercise 3.1 Check Substri

Started on	Thursday, 25 March 2021, 14:05
State	Finished
Completed on	Thursday, 25 March 2021, 14:46
Time taken	41 mins 24 secs
Grade	**63.33** out of 110.00 (**58**%)

Question 1

Incorrect Mark 0.00 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 incorrect.

The correct answer is: Statement coverage

Question 2

Correct Mark 10.00 out of 10.00

Consider the following method:

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

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

Incorrect Mark 0.00 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 incorrect.

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 13.33 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:

- [x] Partitioning
- [x] Boundaries
- [x] Black box
- [ ] Regression
- [ ] Coverage
- [ ] White box
- [x] Integration

Your answer is partially correct.
You have selected too many options.
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

Incorrect | Mark 0.00 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 unit tessting ✗ .

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

Incorrect Mark 0.00 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: MyList p = new MyList(100, null);

The correct answer is: MyList p = this;

Question 10

Correct Mark 10.00 out of 10.00

What is the println result of :

```
MyList3 list = new MyList3(100, null);
list = new MyList3(200, list);
list = new MyList3(300, list);

System.out.println(list.get(0));
```

Answer: 300

The correct answer is: 300

Finish review

Started on	Wednesday, 31 March 2021, 20:38
State	Finished
Completed on	Sunday, 4 April 2021, 15:25
Time taken	3 days 18 hours
Grade	**53.33** out of 110.00 (**48**%)

Question 1

Incorrect Mark 0.00 out of 10.00

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

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

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

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

Incorrect Mark 0.00 out of 10.00

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

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

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

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:

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

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:

```
/**
 * Requires: tiles has length 7 & contains only uppercase letters.
 *           crossings contains only uppercase letters, without duplicates.
 * Effects: Returns a list of words where each word can be made by taking
 *          letters from tiles and at most 1 letter from crossings.
```

```
public static List<String> scrabble(String tiles, String crossings) {
    if (tiles.length() != 7) { throw new RuntimeException(); }
    return new ArrayList<>();
}
```

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

Incorrect Mark 0.00 out of 10.00

Suppose we're working on the method below:

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

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 incorrect.

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

c. restrictions on return values

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

Incorrect | Mark 0.00 out of 10.00

Alice writes the following code:

```
public static int gcd(int a, int b) {
    if (a > b) {
        return gcd(a-b, b);
    } else if (b > a) {
        return gcd(a, b-a);
    }
    return a;
}
```

Bob writes the following test:

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

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 incorrect.

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 :

```
static int find(int[] arr, int val)
    requires: arr[0] == val
    effects:  returns index i such that arr[i] == val
```

Which are the valid test cases for `find` ?

Select one or more:

- [x] `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.

Finish review

Lab 5 Recording | Jump to... | Lab Exercise 5.1 LLDeque EMP

Started on	Thursday, 8 April 2021, 15:49
State	Finished
Completed on	Thursday, 8 April 2021, 17:07
Time taken	1 hour 18 mins
Grade	**70.00** out of 130.00 (**54**%)

Question 1

Correct Mark 20.00 out of 20.00

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

Select one or more:

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

```
public static String nope() {
    return null;            // (1)
}

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

Which line contains a static error ?

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 :

```
public static String nope() {
    return null;            // (1)
}

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

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

Incorrect | Mark 0.00 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 `Point`s 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 incorrect.

The correct answer is: a checked `PathNotFoundException`

Question 5

Incorrect | Mark 0.00 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 incorrect.

The correct answer is: Exception

Question 6

Incorrect | Mark 0.00 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 incorrect.

The correct answer is: RuntimeException

Question 7

Incorrect | Mark 0.00 out of 20.00

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

```
static List<Thing> allTheThings;

static void analyzeEverything() {
    analyzeThings();
}

static void analyzeThings() {
    try {
        for (Thing t : allTheThings) {
            analyzeOneThing(t);
        }
    } catch (AnalysisException ae) {
        return;
    }
}

static void analyzeOneThing(Thing t) throws AnalysisException {
    // ...
    // ... maybe go past the end of a list
    // ...
}
```

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:

- [x] `AnalysisException`
- [ ] `IndexOutOfBoundsException`
- [ ] `NullPointerException`
- [ ] `OutOfMemoryError`

Your answer is incorrect.

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

Question 8

Partially correct | Mark 10.00 out of 20.00

If we want to construct a different object with the same values as the input object, we use a/an other ✗ that performs a/ar

deep copy ✔ instead of a shallow copy.

Question 9

Correct Mark 10.00 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 :

```
if (n % 2 == 0) {
    // your code here

}
```

Do not forget to end it with a semicolon.

Answer: throw new IllegalArgumentException("n must not be even"); ✔

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'.

Finish review

Started on	Friday, 23 April 2021, 14:03
State	Finished
Completed on	Tuesday, 27 April 2021, 15:24
Time taken	4 days 1 hour
Grade	**40.00** out of 110.00 (**36**%)

Question 1

Incorrect Mark 0.00 out of 10.00

Consider the following implementation:

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

and this specification of `find`:

```
static int find(int[] arr, int val)
  requires: nothing
  effects: returns largest index i such that
           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
- [x] none of all others, `findFirst` does satisfy this spec!

Your answer is incorrect.

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

Question 2

Incorrect Mark 0.00 out of 10.00

Consider the following implementation:

```
static int findLast(int[] arr, int val) {
    for (int i = arr.length-1 ; i >= 0; i--) {
```

```
        if (arr[i] == val) return i;
    }
    return -1;
}
```

and this specification of `find`:

```
static int find(int[] arr, int val)
  requires: nothing
  effects: returns largest index i such that
           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 incorrect.

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

Question 3

Incorrect | Mark 0.00 out of 10.00

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

Select one:

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

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

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

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

Your answer is incorrect.

The correct answer is:

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

Question 4

Incorrect | Mark 0.00 out of 10.00

Given this specification:

```
static String join(String delimiter, String[] elements)
  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.
```
effects: returns elements joined together with copies of delimiter, i.e.
         elements[0] + delimiter + elements[1] + delimiter +
         ... + delimiter + elements[elements.length-1]
```

b.
```
effects: returns the result of adding all elements to a
         new StringJoiner(delimiter)
```

c.
```
effects: returns the result of looping through elements and
         alternately appending an element and the delimiter
```

d.
```
effects: returns the result of recursive calls on the elements and
         while concatenating the delimiter
```

Your answer is incorrect.

The correct answer is:

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

Question 5

Incorrect | Mark 0.00 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 incorrect.

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:

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

Incorrect | Mark 0.00 out of 10.00

Here are the `find` specifications from Lecture 8 :

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

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

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

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

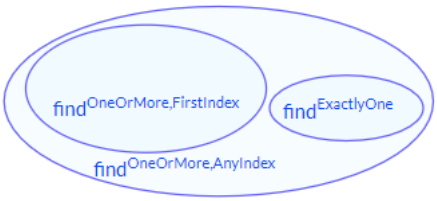
We already know that $\texttt{find}^{\texttt{OneOrMore,FirstIndex}}$ is stronger than $\texttt{find}^{\texttt{OneOrMore,AnyIndex}}$, which is stronger than $\texttt{find}^{\texttt{ExactlyOne}}$.

Where is $\texttt{find}^{\texttt{ExactlyOne}}$ on the diagram ?

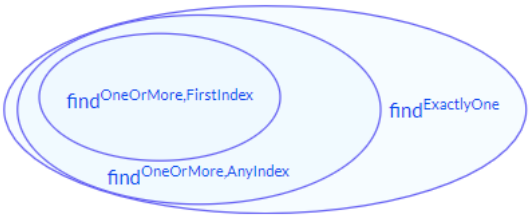
Select one:

a.

b.

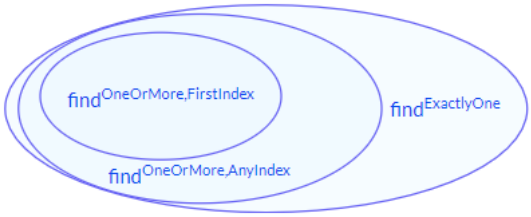


c.



d.

Your answer is incorrect.



The correct answer is:

Question 8

Incorrect Mark 0.00 out of 10.00

Here are the `find` specifications from Lecture 8 :

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

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

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

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

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

How does $\texttt{find}^{\texttt{CanBeMissing}}$ compare to $\texttt{find}^{\texttt{ExactlyOne}}$?

Select one:

a. $\texttt{find}^{\texttt{CanBeMissing}}$ is stronger than $\texttt{find}^{\texttt{ExactlyOne}}$

b. $\texttt{find}^{\texttt{CanBeMissing}}$ is weaker than $\texttt{find}^{\texttt{ExactlyOne}}$

c. $\texttt{find}^{\texttt{CanBeMissing}}$ and $\texttt{find}^{\texttt{ExactlyOne}}$ are incomparable

d. none of the options is correct

Your answer is incorrect.

The correct answer is: $\texttt{find}^{\texttt{CanBeMissing}}$ is stronger than $\texttt{find}^{\texttt{ExactlyOne}}$

Question 9

Correct Mark 10.00 out of 10.00

Here are the `find` specifications from Lecture 8 :

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

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

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

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

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

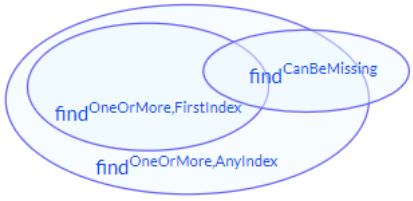
Where is $\texttt{find}^{\texttt{CanBeMissing}}$ on the diagram ?

Select one:

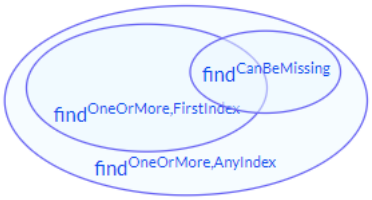
a.

b.

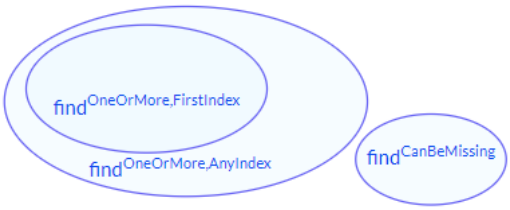
c.



d.



e.



Your answer is correct.

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:

```
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")

Finish review

Lab 8 Recording Jump to... Lab Exercise 8.1 ARDeque EMP

Started on	Tuesday, 4 May 2021, 13:42
State	Finished
Completed on	Tuesday, 4 May 2021, 13:44
Time taken	1 min 53 secs
Marks	60.00/150.00
Grade	**64.00** out of 160.00 (**40**%)

Question 1

Incorrect Mark 0.00 out of 10.00

Consider the following code, executed in order:

```
char text0 = 'a';
final char text1 = vowel0;

String text2 = text1 + "eiou";
final String text3 = text2;

char[] text4 = new char[] { text0, 'e', 'i', 'o', 'u' };
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 incorrect.

The correct answer is: text2 = text3;

Question 2

Correct Mark 10.00 out of 10.00

Consider the following code, executed in order:

```
char text0 = 'a';
final char text1 = vowel0;

String text2 = text1 + "eiou";
final String text3 = text2;
```

```
char[] text4 = new char[] { text0, 'e', 'i', 'o', 'u' };
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

Incorrect Mark 0.00 out of 10.00

Consider this (incomplete) method:

```
/**
 * Solves quadratic equation ax^2 + bx + c = 0.
 *
 * @param a quadratic coefficient, requires a != 0
 * @param b linear coefficient
 * @param c constant term
 * @return a list of the real roots of the equation
 */
public static List<Double> quadraticRoots(final int a, final int b, final int c) {
    List<Double> roots = new ArrayList<Double>();
    // A
    ... // compute roots
    // B
    return roots;
}
```

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 incorrect.

The correct answer is: assert a != 0;

Question 4

Incorrect Mark 0.00 out of 10.00

Consider this (incomplete) method:

```
/**
 * Solves quadratic equation ax^2 + bx + c = 0.
 *
 * @param a quadratic coefficient, requires a != 0
 * @param b linear coefficient
 * @param c constant term
 * @return a list of the real roots of the equation
 */
public static List<Double> quadraticRoots(final int a, final int b, final int c) {
    List<Double> roots = new ArrayList<Double>();
    // A
    ... // compute roots
    // B
    return roots;
}
```

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 incorrect.

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 :

```
class Apartment {

    Apartment(String newAddress) {
        this.address = newAddress;
        this.roommates = new HashSet<Person>();
    }

    String getAddress() {
        return address;
    }

    void addRoommate(Person newRoommate) {
        roommates.add(newRoommate);
        if (roommates.size() > MAXIMUM_OCCUPANCY) {
            roommates.remove(newRoommate);
            throw new TooManyPeopleException();
        }
    }

    int getMaximumOccupancy() {
        return MAXIMUM_OCCUPANCY;
    }
}
```

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 :

```
class Apartment {

    Apartment(String newAddress) {
        this.address = newAddress;
        this.roommates = new HashSet<Person>();
    }

    String getAddress() {
        return address;
    }

    void addRoommate(Person newRoommate) {
        roommates.add(newRoommate);
        if (roommates.size() > MAXIMUM_OCCUPANCY) {
            roommates.remove(newRoommate);
            throw new TooManyPeopleException();
        }
    }

    int getMaximumOccupancy() {
        return MAXIMUM_OCCUPANCY;
    }
}
```

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;

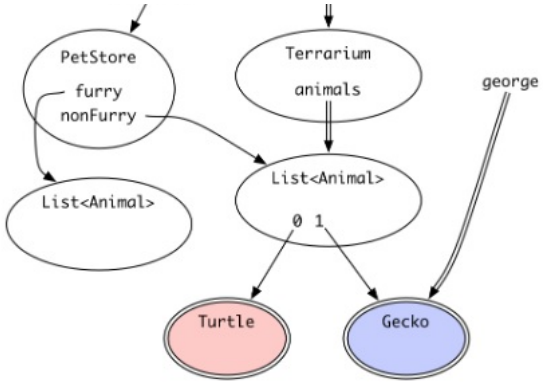
Your answer is correct.

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

Question 7

Incorrect Mark 0.00 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

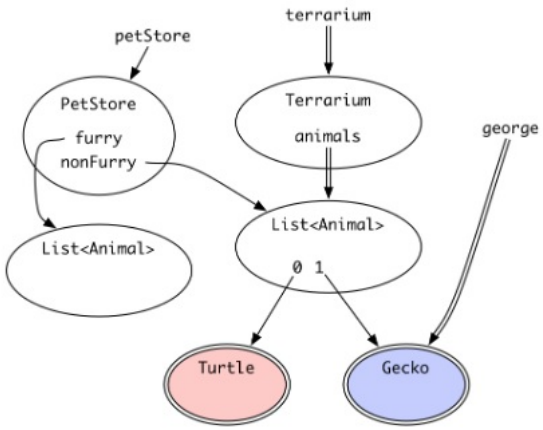
Your answer is incorrect.

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

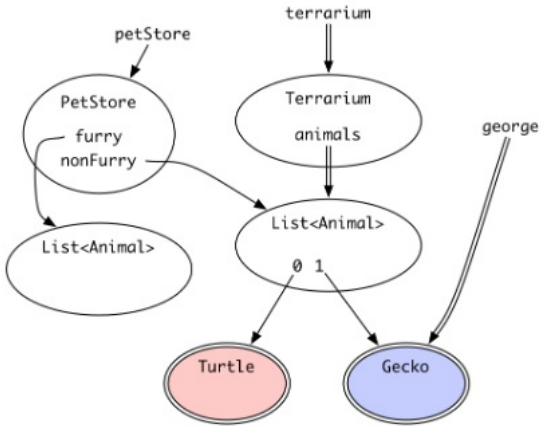
Your answer is incorrect.

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 th 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:

```
public class MyIterator {
```

```
    private final ArrayList<String> list;
    private int index;

    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        index++;
        return element;
    }
}
```

What is the type of the input to `next`?

Select one:

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

Your answer is incorrect.

The correct answer is: MyIterator

Question 11

Correct | Mark 10.00 out of 10.00

Consider `MyIterator`'s `next` method:

```
public class MyIterator {

    private final ArrayList<String> list;
    private int index;

    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        index++;
        return element;
    }
}
```

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:

```
public class MyIterator {

    private final ArrayList<String> list;
    private int index;

    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        ++index;
        return element;
    }
}
```

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

Which input to `next` is constrained by the precondition?

Select one:

a. this

b. list

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:

```
public class MyIterator {

    private final ArrayList<String> list;
    private int index;
```

```
    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        ++index;
        return element;
    }
}
```

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

Incorrect | Mark 0.00 out of 10.00

Consider `MyIterator`'s `next` method:

```
public class MyIterator {

    private final ArrayList<String> list;
    private int index;

    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        ++index;
        return element;
    }
}
```

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 incorrect.

The correct answer is: the return value

Question 15

Correct | Mark 10.00 out of 10.00

Consider `MyIterator`'s `next` method:

```
public class MyIterator {

    private final ArrayList<String> list;
    private int index;

    ...

    /**
     * Get the next element of the list.
     * Requires: hasNext() returns true.
     * Modifies: this iterator to advance it to the element
     *           following the returned element.
     * @return next element of the list
     */
    public String next() {
        final String element = list.get(index);
        ++index;
        return element;
    }
}
```

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

Finish review

Lab 9 Recording | Jump to... | Lab Exercise 9.1 ARDequeItera

Started on	Sunday, 9 May 2021, 15:09
State	Finished
Completed on	Sunday, 9 May 2021, 16:41
Time taken	1 hour 31 mins
Grade	**50.00** out of 150.00 (**33**%)

Question 1

Incorrect Mark 0.00 out of 10.00

Consider an abstract data type `Bool`.
The type has the following operations:

```
true : Bool
false : Bool

and : Bool × Bool → Bool
or : Bool × Bool → Bool
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

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 :

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 incorrect.

The correct answer is: creator

Question 3

Correct Mark 10.00 out of 10.00

The method below is an c

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-java.math.BigInteger-

Select one:

a. creator

b. producer

c. mutator

d. observer

Your answer is correct.

The correct answer is: producer

Question 4

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 :

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 incorrect.

The correct answer is: mutator

Question 5

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 :

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 correct.

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

b. producer

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.

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family, sorted from oldest to youngest, with no duplicates.
    public List<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return people;
    }
}
```

Here is a client of this abstract data type:

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

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:

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family
    public Set<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return new ArrayList<>(people);
    }
}
```

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 answ

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.

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family, sorted from oldest to youngest, with no duplicates.
    public List<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return people;
    }
}
```

Here is a client of this abstract data type:

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

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:

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family
    public Set<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return new ArrayList<>(people);
    }
}
```

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 runtim

e. `client2` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answ

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.

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family, sorted from oldest to youngest, with no duplicates.
    public List<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return people;
    }
}
```

Here is a client of this abstract data type:

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

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:

```
/**
 * Represents a family that lives in a household together.
 * A family always has at least one person in it.
 * Families are mutable.
 */
class Family {
    // the people in the family
    public Set<Person> people;

    /**
     * @return a list containing all the members of the family, with no duplicates.
     */
    public List<Person> getMembers() {
        return new ArrayList<>(people);
    }
}
```

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 runtim

e. `client3` depends on `Family`'s representation, and the dependency would not be caught but would (luckily) still produce the same answ

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  /**
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  /**
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

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

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

Incorrect Mark 0.00 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 incorrect.

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

Finish review

Lab 10 Recording

Jump to...

Lab Exercise 10.1 ARDeque DE

Started on	Friday, 14 May 2021, 15:12
State	Finished
Completed on	Friday, 14 May 2021, 20:53
Time taken	5 hours 41 mins
Grade	**17.50** out of 150.00 (**12**%)

Question 1

Incorrect | Mark 0.00 out of 10.00

Consider the following problematic datatype:

```
/** Represents an immutable right triangle. */
class RightTriangle {
/*A*/   private double[] sides;

        // sides[0] and sides[1] are the two legs,
        // and sides[2] is the hypotenuse, so declare it to avoid having a
        // magic number in the code:
/*B*/   public static final int HYPOTENUSE = 2;

        /** Mak
         * @par
         * @param hypotenuse    the hypotenuse of the triangle.
 *C*     *         Requires hypotenuse^2 = legA^2 + legB^2
         *           (within the error tolerance of double arithmetic)
         */
        public RightTriangle(double legA, double legB, double hypotenuse) {
/*D*/       this.sides = new double[] { legA, legB, hypotenuse };
        }

        /** Get all the sides of the triangle.
         *  @return three-element array with the triangle's side lengths
         */
        public double[] getAllSides() {
            return sides;
        }

        /** @return length of the triangle's hypotenuse */
        public double getHypotenuse() {
            return sides[HYPOTENUSE];
30.     }

        /** @param factor to multiply the sides by
         *  @return a triangle made from this triangle by
         *  multiplies all side lengths by factor.
         */
        public RightTriangle scale(double factor) {
            return new RightTriangle (sides[0]*factor, sides[1]*factor, sides[2]*factor);
        }

        /** @return a regular triangle made from this triangle.
         *  A regular right triangle is one in which
         *  both legs have the same length.
         */
        public RightTriangle regularize() {
            double bigLeg = Math.max(side[0], side[1]);
            return new RightTriangle (bigLeg, bigLeg, side[2]);
        }

    }
```

XJTLU LEARNING MALL ONLINE Need help? English (en)

Which of the following statements are true?

Select one:

a. The line marked /*A*/ is a problem for rep exposure because arrays are mutable.

b. The line marked /*B*/ is a problem for representation independence because it reveals how the sides array is organized.

c. The line marked *C* is a problem because creator operations should not have preconditions.

d. The line marked /*D*/ is a problem because it puts legA, legB, and hypotenuse into the rep without doing a defensive copy first.

Your answer is incorrect.

The correct answer is: The line marked /*B*/ is a problem for representation independence because it reveals how the sides array is organized.

Question 2

Incorrect Mark 0.00 out of 10.00

Which of the following should **not** be known (visible and documented) to **the client** of an abstract data type?

Select one:

a. rep invariant

b. abstract value space

c. creators

d. observers

Your answer is incorrect.

The correct answer is: rep invariant

Question 3

Incorrect Mark 0.00 out of 10.00

Which of the following should be known (visible and documented) to **the maintainer** of an abstract data type?

Select one:

a. all of the options

b. abstract value space

c. creators

d. observers

e. abstraction function

f. rep

g. rep invariant

Your answer is incorrect.

The correct answer is: all of the options

Question 4

Correct | Mark 10.00 out of 10.00

Suppose C is an abstract data type whose representation has two String fields:

```
class C {
    private String s;
    private String t;
    ...
}
```

Assuming you don't know anything about C's abstraction, which of the following might be part of a rep invariant for C?

Select one:

- a. s.length() == t.length()
- b. s represents a set of characters
- c. C's observers
- d. s + t

Your answer is correct.

The correct answer is: s.length() == t.length()

Question 5

Incorrect | Mark 0.00 out of 10.00

Suppose we are implementing CharSet with the following rep:

```
public class CharSet {
    private String s;
    ...
}
```

But we neglect to write down the abstraction function (AF) and rep invariant (RI). Here are four possible AF/RI pairs, which were also mentione the lecture.

SortedRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s[0] < s[1] < ... < s[s.length()-1]
```

SortedRangeRep:

```
// AF: represents the union of the ranges {s[i]...s[i+1]} for each adjacent pair of characters in s
// RI: s.length is even, and s[0] < s[1] < ... < s[s.length()-1]
```

NoRepeatsRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s contains no character more than once
```

AnyRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: true
```

Which possible AF/RI pairs are consistent with this programmer's implementation of `add()`?

```
/**
 * Modifies this set by adding c to the set.
 * @param c character to add
 */
public void add(char c) {
    s = s + c;
}
```

Select one:

a. SortedRep

b. SortedRangeRep

c. NoRepeatsRep

d. AnyRep

Your answer is incorrect.

The correct answer is: AnyRep

Question 6

Incorrect | Mark 0.00 out of 10.00

Suppose we are implementing CharSet with the following rep:

```
public class CharSet {
    private String s;
    ...
}
```

But we neglect to write down the abstraction function (AF) and rep invariant (RI). Here are four possible AF/RI pairs, which were also mentioned the lecture.

SortedRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s[0] < s[1] < ... < s[s.length()-1]
```

SortedRangeRep:

```
// AF: represents the union of the ranges [s[i]...s[i+1]] for each adjacent pair of characters in s
// RI: s.length is even, and s[0] < s[1] < ... < s[s.length()-1]
```

NoRepeatsRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s contains no character more than once
```

AnyRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: true
```

Which possible AF/RI pairs are consistent with this programmer's implementation of `remove()`?

```
/**
 * Modifies this set by removing c, if found.
 * If c is not found in the set, has no effect.
 * @param c character to remove
 */
public void remove(char c) {
    int position = s.indexOf(c);
    if (position >= 0) {
        s = s.substring(0, position) + s.substring(position+1, s.length());
    }
}
```

Select one or more:

i. SortedRep

ii. SortedRangeRep

iii. NoRepeatsRep

iv. AnyRep

Your answer is incorrect.

The correct answers are: SortedRep, NoRepeatsRep

Question 7

Incorrect | Mark 0.00 out of 10.00

Suppose we are implementing CharSet with the following rep:

```
public class CharSet {
    private String s;
    ...
}
```

But we neglect to write down the abstraction function (AF) and rep invariant (RI). Here are four possible AF/RI pairs, which were also mentioned the lecture.

SortedRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s[0] < s[1] < ... < s[s.length()-1]
```

SortedRangeRep:

```
// AF: represents the union of the ranges {s[i]...s[i+1]} for each adjacent pair of characters in s
// RI: s.length is even, and s[0] < s[1] < ... < s[s.length()-1]
```

NoRepeatsRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: s contains no character more than once
```

AnyRep:

```
// AF: {s[i] | 0 <= i < s.length()}
// RI: true
```

Finally, which possible AF/RI pairs are consistent with this programmer's implementation of `contains()`?

```
/**
 * Test for membership.
 * @param c a character
 * @return true iff this set contains c
 */
public boolean contains(char c) {
    for (int i = 0; i < s.length(); i += 2) {
        char low = s.charAt(i);
        char high = s.charAt(i+1);
        if (low <= c && c <= high) {
            return true;
        }
    }
    return false;
}
```

Select one:

- a. SortedRep
- b. SortedRangeRep
- c. NoRepeatsRep
- d. AnyRep

Your answer is incorrect.

The correct answer is: SortedRangeRep

Question 8

Incorrect | Mark 0.00 out of 10.00

Consider this ADT:

```
public class Duration {
    private final int mins;
    private final int secs;
    // rep invariant:
    //    mins >= 0, secs >= 0
    // abstraction function:
    //    represents a span of time of mins minutes and secs seconds

    /** Make a duration lasting for m minutes and s seconds. */
    public Duration(int m, int s) {
        mins = m; secs = s;
    }
    /** @return length of this duration in seconds */
    public long getLength() {
        return mins*60 + secs;
    }
}
```

and these objects created from it:

```
Duration d1 = new Duration (1, 2);
Duration d2 = new Duration (1, 3);
Duration d3 = new Duration (0, 62);
Duration d4 = new Duration (1, 2);
```

Using the abstraction-function notion of equality, which of the following would be considered **equal to** `d1`?

Select one or more:

- [ ] i. d1
- [x] ii. d2
- [ ] iii. d3
- [ ] iv. d4

Your answer is incorrect.

The correct answers are: d1, d3, d4

Question 9

Partially correct | Mark 3.33 out of 10.00

Consider this ADT:

```
public class Duration {
    private final int mins;
    private final int secs;
    // rep invariant:
    //    mins >= 0, secs >= 0
    // abstraction function:
    //    represents a span of time of mins minutes and secs seconds

    /** Make a duration lasting for m minutes and s seconds. */
    public Duration(int m, int s) {
        mins = m; secs = s;
    }
    /** @return length of this duration in seconds */
    public long getLength() {
        return mins*60 + secs;
    }
}
```

and these objects created from it:

```
Duration d1 = new Duration (1, 2);
Duration d2 = new Duration (1, 3);
Duration d3 = new Duration (0, 62);
```

```
Duration d4 = new Duration (1, 2);
```

Using the observational notion of equality, which of the following would be considered **equal to** `d1`?

Select one or more:

- i. d1
- ii. d2
- iii. d3
- iv. d4

Your answer is partially correct.
You have correctly selected 1.
The correct answers are: d1, d3, d4

Question 10

Incorrect | Mark 0.00 out of 10.00

Consider the latest implementation of `Duration` in the lecture:

```
public class Duration {
    private final int mins;
    private final int secs;
    // rep invariant:
    //   mins >= 0, secs >= 0
    // abstraction function:
    //   represents a span of time of mins minutes and secs seconds

    /** Make a duration lasting for m minutes and s seconds. */
    public Duration(int m, int s) {
        mins = m; secs = s;
    }
    /** @return length of this duration in seconds */
    public long getLength() {
        return mins*60 + secs;
    }

    private static final int CLOCK_SKEW = 5; // seconds

    @Override
    public boolean equals (Object thatObject) {
        if (!(thatObject instanceof Duration)) return false;
        Duration thatDuration = (Duration) thatObject;
        return Math.abs(this.getLength() - thatDuration.getLength()) <= CLOCK_SKEW;
    }
}
```

Suppose these Duration objects are created:

```
Duration d_0_60 = new Duration(0, 60);
Duration d_1_00 = new Duration(1, 0);
Duration d_0_57 = new Duration(0, 57);
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 incorrect.

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 11

Incorrect Mark 0.00 out of 10.00

Consider the latest implementation of `Duration` in the lecture:

```
public class Duration {
    private final int mins;
    private final int secs;
    // rep invariant:
    //    mins >= 0, secs >= 0
    // abstraction function:
    //    represents a span of time of mins minutes and secs seconds

    /** Make a duration lasting for m minutes and s seconds. */
    public Duration(int m, int s) {
        mins = m; secs = s;
    }
    /** @return length of this duration in seconds */
    public long getLength() {
        return mins*60 + secs;
    }

    private static final int CLOCK_SKEW = 5; // seconds

    @Override
    public boolean equals (Object thatObject) {
        if (!(thatObject instanceof Duration)) return false;
        Duration thatDuration = (Duration) thatObject;
        return Math.abs(this.getLength() - thatDuration.getLength()) <= CLOCK_SKEW;
    }
}
```

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 12

Incorrect Mark 0.00 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 incorrect.
The correct answer is: 1 object

Question 13

Partially correct | Mark 0.83 out of 10.00

Suppose Bag<E> is a mutable ADT representing what is often called a *multiset*, an unordered collection of objects where an object can occur m than once. It has the following operations:

```
/** make an empty bag */
public Bag<E>()

/** modify this bag by adding an occurence of e, and return this bag */
public Bag<E> add(E e)

/** modify this bag by removing an occurence of e (if any), and return this bag */
public Bag<E> remove(E e)

/** return number of times e occurs in this bag */
public int count(E e)
```

Suppose we run this code:

```
Bag<String> b1 = new Bag<>().add("a").add("b");
Bag<String> b2 = new Bag<>().add("a").add("b");
Bag<String> b3 = b1.remove("b");
Bag<String> b4 = new Bag<>().add("b").add("a"); // swap!
```

Which of the following expression is **true** ?

Select one or more:

- i. b1.count("a") == 1
- ii. b1.count("b") == 1
- iii. b2.count("a") == 1
- iv. b2.count("b") == 1
- v. b3.count("a") == 1
- vi. b3.count("b") == 1
- vii. b4.count("a") == 1
- viii. b4.count("b") == 1

Your answer is partially correct.
You have correctly selected 2.
The correct answers are: b1.count("a") == 1, b2.count("a") == 1, b2.count("b") == 1, b3.count("a") == 1, b4.count("a") == 1, b4.count("b") == 1

Question 14

Incorrect | Mark 0.00 out of 10.00

Suppose Bag<E> is a mutable ADT representing what is often called a *multiset*, an unordered collection of objects where an object can occur m than once. It has the following operations:

```
/** make an empty bag */
public Bag<E>()

/** modify this bag by adding an occurence of e, and return this bag */
public Bag<E> add(E e)

/** modify this bag by removing an occurence of e (if any), and return this bag */
public Bag<E> remove(E e)

/** return number of times e occurs in this bag */
public int count(E e)
```

Suppose we run this code:

```
Bag<String> b1 = new Bag<>().add("a").add("b");
Bag<String> b2 = new Bag<>().add("a").add("b");
Bag<String> b3 = b1.remove("b");
Bag<String> b4 = new Bag<>().add("b").add("a"); // swap!
```

If Bag is implemented with **behavioral** equality, which of the following expression is **true** ?

Select one or more:

- [x] i. b1.equals(b2)
- [ ] ii. b1.equals(b3)
- [ ] iii. b1.equals(b4)
- [x] iv. b2.equals(b3)
- [ ] v. b2.equals(b4)
- [ ] vi. b3.equals(b1)

Your answer is incorrect.

The correct answers are: b1.equals(b3), b3.equals(b1)

Question 15

Partially correct | Mark 3.33 out of 10.00

Suppose Bag<E> is a mutable ADT representing what is often called a *multiset*, an unordered collection of objects where an object can occur m than once. It has the following operations:

```
/** make an empty bag */
public Bag<E>()

/** modify this bag by adding an occurence of e, and return this bag */
public Bag<E> add(E e)

/** modify this bag by removing an occurence of e (if any), and return this bag */
public Bag<E> remove(E e)

/** return number of times e occurs in this bag */
public int count(E e)
```

Suppose we run this code:

```
Bag<String> b1 = new Bag<>().add("a").add("b");
Bag<String> b2 = new Bag<>().add("a").add("b");
Bag<String> b3 = b1.remove("b");
Bag<String> b4 = new Bag<>().add("b").add("a"); // swap!
```

If Bag is implemented with **observational equality**, which of the following expression is **true** ?

Select one or more:

- [ ] i. b1.equals(b2)
- [x] ii. b1.equals(b3)
- [ ] iii. b1.equals(b4)
- [x] iv. b2.equals(b3)
- [x] v. b2.equals(b4)
- [ ] vi. b3.equals(b1)

Your answer is partially correct.

You have correctly selected 2.
The correct answers are: b1.equals(b3),
b2.equals(b4),
b3.equals(b1)

Finish review

Lab 11 Recording Jump to... Lab Exercise 11.1 Duration CO

Started on	Tuesday, 25 May 2021, 12:41
State	Finished
Completed on	Tuesday, 25 May 2021, 12:49
Time taken	7 mins 37 secs
Grade	**75.00** out of 140.00 (**54**%)

Question 1

Correct Mark 10.00 out of 10.00

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

```
Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();
a.put("c", 130); // put ints into the map
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 `String`s and the values are `Integer`s. 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

Correct Mark 10.00 out of 10.00

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

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

Your answer is correct.

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

Correct Mark 10.00 out of 10.00

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

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

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

```
int i = a.get("c");
int j = b.get("c");
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 correct.

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

Question 4

Partially correct | Mark 5.00 out of 10.00

For this code that starts a thread:

```
new Thread(new Runnable() {
    public void run() {
        System.out.println("Hello! It's me, a thread!");
    }
}).start();
```

Put the following events in the order that they occur.

1. start() is called ✗
2. a Thread object is created ✓
3. run() is called ✗
4. a Runnable object is created ✗
5. "Hello! It's me, a thread!" is printed ✓
6. run() returns ✓

Your answer is partially correct.

The expression `new Runnable() { ... }` creates a new object that implements `Runnable`, which will be passed as a parameter to `new Thread()`. Note especially that the code inside the anonymous class *is not executed yet*. It won't be executed until its `run()` method is called.

Once we have the `Runnable` object, the next thing that happens is the call to `new Thread()`, which creates a new `Thread` object.

Then `start()` is called on that new `Thread` object.

The thread then starts, and `Thread.start()` calls `run()` on the `Runnable` object.

Inside the body of `run()`, the `println` statement executes.

Finally, the `run()` method returns, and the thread finishes.

You have correctly selected 3.
The correct answer is:
For this code that starts a thread:

```
new Thread(new Runnable() {
    public void run() {
        System.out.println("Hello! It's me, a thread!");
    }
}).start();
```

Put the following events in the order that they occur.

1. [a Runnable object is created]
2. [a Thread object is created]
3. [start() is called]
4. [run() is called]
5. ["Hello! It's me, a thread!" is printed]
6. [run() returns]

Question 5

Incorrect | Mark 0.00 out of 10.00

When you run a Java program (for example, using the Run button in IntelliJ), how many processes and threads are created at first?

Select one:

- a. one process and one thread
- b. one process and zero thread
- c. zero process and one thread
- d. one process for each class, and one thread for each class in the program
- e. one process, and one thread for each class in the program
- f. one process for each class in the program, and one thread
- g. zero process and zero thread

Your answer is incorrect.

The correct answer is: one process and one thread

Question 6

Incorrect | Mark 0.00 out of 10.00

Suppose we run main in this program, which contains bugs:

```
public class Moirai {
    public static void main(String[] args) {
        Thread clotho = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        clotho.start();
        new Thread(new Runnable() {
            public void run() { System.out.println("measuring"); };
        }).start();
        new Thread(new Runnable() {
            public void run() { System.out.println("cutting"); };
        });
    }
}
```

How many new Thread objects are created?

Select one:

- a. 3
- b. 2
- c. 1
- d. 0
- e. 4
- f. 5
- g. 6

Your answer is incorrect.

One is assigned to variable `clotho`. The other two are not assigned to a variable.

The correct answer is: 3

Question 7

Correct | Mark 10.00 out of 10.00

Suppose we run main in this program, which contains bugs:

```
public class Moirai {
    public static void main(String[] args) {
        Thread clotho = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        clotho.start();
        new Thread(new Runnable() {
            public void run() { System.out.println("measuring"); };
        }).start();
        new Thread(new Runnable() {
            public void run() { System.out.println("cutting"); };
        });
    }
}
```

How many new threads are run?

Select one:

- a. 3
- b. 2
- c. 1
- d. 0
- e. 4
- f. 5
- g. 6

Your answer is correct.

The code calls `start` on the first two threads. But the third thread is never `start`ed, so it will not run.

The correct answer is: 2

Question 8

Incorrect | Mark 0.00 out of 10.00

Suppose we run main in this program, which contains bugs:

```
public class Moirai {
    public static void main(String[] args) {
        Thread clotho = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        clotho.start();
        new Thread(new Runnable() {
            public void run() { System.out.println("measuring"); };
        }).start();
        new Thread(new Runnable() {
```

```
            public void run() { System.out.println("cutting"); };
        });
    }
}
```

What is the maximum number of threads that might be running at the same time?

Select one:

- a. 3
- b. 2
- c. 1
- d. 0
- e. 4
- f. 5
- g. 6

Your answer is incorrect.

The initial thread running `main` plus the two new threads that were started. The reason we have to say "might" here is because different interleaving may mean that we don't always reach this maximum; for example, the first new thread might finish running before the second one even starts.

The correct answer is: 3

Question 9

Incorrect | Mark 0.00 out of 10.00

Suppose we run main in this program, which demonstrates two common bugs:

```
public class Parcae {
    public static void main(String[] args) {
        Thread nona = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        nona.run();
        Runnable decima = new Runnable() {
            public void run() { System.out.println("measuring"); };
        };
        decima.run();
        // ...
    }
}
```

How many new Thread objects are created (not counting the main thread) ?

Select one:

- a. 3
- b. 2
- c. 1
- d. 0
- e. 4
- f. 5
- g. 6

Your answer is incorrect.

We create only one `Thread`, assigned to variable `nona`.

The correct answer is: 1

Question 10

Incorrect | Mark 0.00 out of 10.00

Suppose we run main in this program, which demonstrates two common bugs:

```
public class Parcae {
    public static void main(String[] args) {
        Thread nona = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        nona.run();
        Runnable decima = new Runnable() {
            public void run() { System.out.println("measuring"); };
        };
        decima.run();
        // ...
    }
}
```

How many new threads are run?

Select one:

- a. 3
- b. 2
- c. 1
- d. 0
- e. 4
- f. 5
- g. 6

Your answer is incorrect.

The line `nona.run()` is a bug: calling `Thread.run()` does not run the code in a new concurrent thread. It uses the same thread, the initial thread running `main`.

And we call `run()` on `Runnable decima`, which also uses the same thread. Perhaps the author meant to create a new `Thread` with that `Runnable` instead of running it directly.

Never call `run()` on a `Thread`, or on a `Runnable` that you created for a thread. Instead, always make a `new Thread()` with an instance of your `Runnable`, and call `start()` on the thread to start it. `Thread` will take care of calling `run()` on your `Runnable` from the new thread.

The correct answer is: 0

Question 11

Partially correct | Mark 2.50 out of 10.00

Suppose we run main in this program, which contains bugs:

```
public class Moirai {
    public static void main(String[] args) {
        Thread clotho = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        clotho.start();
        new Thread(new Runnable() {
```

```
            public void run() { System.out.println("measuring"); };
        }).start();
        new Thread(new Runnable() {
            public void run() { System.out.println("cutting"); };
        });
    }
}
```

Which of the following is a possible output from this program?

Select one or more:

- [ ] i. measuring
- [ ] ii. measuring
  spinning
- [ ] iii. spinning
- [x] iv. spinning
  measuring
  cutting
- [ ] v. cutting
- [ ] vi. spinning
  cutting
- [x] vii. spinning
  measuring

Your answer is partially correct.

The third thread is never started, so `cutting` will never be printed.

The order of the other outputs depends on whether the first thread runs `println` before or after the second.

Note that `main()` may very well return while the two threads it created are still running. This ends the main thread of the program, but it does not stop the entire process. In Java, the process continues running until all running threads have exited, unless `System.exit()` is called to force the proces to exit.

You have correctly selected 1.
The correct answers are: measuring
spinning, spinning
measuring

Question 12

Correct | Mark 10.00 out of 10.00

Suppose we run main in this program, which demonstrates two common bugs:

```
public class Parcae {
    public static void main(String[] args) {
        Thread nona = new Thread(new Runnable() {
            public void run() { System.out.println("spinning"); };
        });
        nona.run();
        Runnable decima = new Runnable() {
            public void run() { System.out.println("measuring"); };
        };
        decima.run();
        // ...
    }
}
```

Which of the following is a possible output from this program?

Select one or more:

- [ ] i. measuring

ii. spinning

iii. measuring
spinning

iv. spinning
measuring

Your answer is correct.

There is only one thread running in this program, and only one possible output. Both `nona.run()` and `decima.run()` run their code in the current thread, the initial thread running `main`.

The correct answer is: spinning
measuring

Question 13

Correct | Mark 10.00 out of 10.00

Consider the following code:

```
private static int x = 1;

public static void methodA() {
  x *= 2;
  x *= 3;
}

public static void methodB() {
  x *= 5;
}
```

Suppose methodA and methodB run **sequentially**, i.e. first one and then the other.

What is the final value of x?

Select one:

a. 1

b. 2

c. 5

d. 30

e. 6

f. 10

g. 150

Your answer is correct.

If `methodA` runs first, then it sets `x` to 1×2×3 = 6, and then `methodB` runs and sets `x` to 6×5 = 30. Since multiplication is commutative, we get the same result if `methodB` runs before `methodA`.

The correct answer is: 30

Question 14

Partially correct | Mark 7.50 out of 10.00

Consider the following code:

```
private static int

public static void
  x *= 2;
  x *= 3;
}

public static void methodB() {
  x *= 5;
}
```


Now suppose methodA and methodB run **concurrently**, so that their instructions might **interleave** arbitrarily, exhibiting **a race condition**.

Which of the following are ***all*** possible final values of x?

Select one or more:

- [ ] i. 1
- [ ] ii. 2
- [x] iii. 5
- [x] iv. 6
- [ ] v. 10
- [x] vi. 30
- [ ] vii. 150

Your answer is partially correct.

30 is still a possible final value, because `methodA` might manage to run completely before `methodB` even starts. So the sequential execution (one before the other) is still always possible when two modules run concurrently. But it's not the only possible result.

One way to get 5 as the final result is if `methodB` first reads the value of `x` (1), and then `methodA` interleaves, running completely (setting `x` to 6). Then `methodB` finishes its computation using the original value of `x` that it read (1), setting `x` to 5 and overwriting `methodA`'s effect.

A similar argument allows us to get 6 as the final result: `methodA` first reads the value of `x`, then `methodB` runs completely, and then `methodA` computes `x` = 6 and overwrites what `methodB` did.

To get 10 as the final result, `methodA` might first successfully multiply by 2 and store that back to `x`. `methodB` then starts, reading the value of x (now 2) and starting its multiplication. Then `methodA` runs again, multiplying `x` by 3 and storing it back (6). Finally `methodB` finishes, using a stale value of `x`, and storing back 2×5 = 10, overwriting the result of `methodA`'s second multiplication.

We can't get 1 as a result, because `methodA` and `methodB` will compute *some* new value of `x` and store it back.

We also can't get 2 as a result, because the final value of `x` will come from either `methodA` or `methodB`'s last computation, which is either a multiplication by 3 or a multiplication by 5.

We also can't get 150, because none of the multiplications will produce a value that large.

You have correctly selected 3.
The correct answers are: 5, 6, 10, 30

Finish review

Lab 12 Recording Jump to... Lab Exercise 12.1 HASet SIZEC

Started on	Wednesday, 26 May 2021, 15:50
State	Finished
Completed on	Tuesday, 1 June 2021, 13:21
Time taken	5 days 21 hours
Grade	**30.83** out of 130.00 (**24**%)

Question 1

Partially correct | Mark 2.50 out of 10.00

In the `main` method of `Factorial` class in the lecture:

```
public static void main(String[] args) {
    new Thread(new Runnable() { // create a thread using an
        public void run() {     // anonymous Runnable
            computeFact(99);
        }
    }).start();
    computeFact(100);
}
```

Which of the following are possible interleavings ?

Select one or more:

- [ ] i. The call to computeFact(99) finishes before the call to computeFact(100) starts
- [x] ii. The call to computeFact(100) starts before the call to computeFact(99) starts
- [ ] iii. The call to computeFact(99) starts before the call to computeFact(100) starts
- [ ] iv. The call to computeFact(100) finishes before the call to computeFact(99) starts

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: The call to computeFact(99) finishes before the call to computeFact(100) starts, The call to computeFact(100) starts before the call to computeFact(99) starts, The call to computeFact(99) starts before the call to computeFact(100) starts, The call to computeFact(100) finishes before the call to computeFact(99) starts

Question 2

Incorrect | Mark 0.00 out of 10.00

Now consider this different `main` that runs three factorial computations:

```
public static void main(String[] args) {
    computeFact(98);
    Thread t = new Thread(new Runnable() {
        public void run() {
            computeFact(99);
```

```
        });
        computeFact(100);
        t.start();
    }
```

Which of the following are possible ordering of events?

Select one or more:

- i. The call to computeFact(98) starts before the call to computeFact(99) starts
- ii. The call to computeFact(99) starts before the call to computeFact(100) starts
- iii. The call to computeFact(99) finishes before the call to computeFact(98) starts
- iv. The call to computeFact(100) finishes before the call to computeFact(99) starts

Your answer is incorrect.
In this code, `computeFact(98)` first executes serially, from start to end. Then a thread for `computeFact(99)` is created *but not started yet*.
Then `computeFact(100)` executes serially from start to end, and finally thread `t` is started to finally run `computeFact(99)`. The `main` method then returns and the main thread finishes running, but the `t` thread continues until it finishes `computeFact(99)` and returns from its `run` method.

The effect of this code is actually three serialized calls: `computeFact(98)` followed by `computeFact(100)` followed by `computeFact(99)` So the first and fourth ordering above *always* happen, but the second and third *never* happen.

Be careful to understand the difference between creating a `Thread` object and `start`ing it.

The correct answers are: The call to computeFact(98) starts before the call to computeFact(99) starts, The call to computeFact(100) finishes before the call to computeFact(99) starts

Question 3

Partially correct | Mark 3.33 out of 10.00

Here's part of the pinball simulator example from the lecture:

```
public class PinballSimulator {

    private static PinballSimulator simulator = null;

    // invariant: there should never be more than one PinballSimulator object created

    public static PinballSimulator getInstance() {
/* 1 */     if (simulator == null) {
/* 2 */         simulator = new PinballSimulator();
            }
/* 3 */     return simulator;
    }
}
```

We want to have the invariant that only one simulator object is created.

Suppose two threads are running `getInstance()`.

One thread is about to execute one of the numbered lines above; the other thread is about to execute the other.

For each pair of possible line numbers, is it possible the invariant will be violated?

Select one or more:

- i. about to execute lines 1 and 3
- ii. about to execute lines 1 and 2
- iii. about to execute lines 1 and 1

Your answer is partially correct.
You have correctly selected 1

The correct answers are: about to execute lines 1 and 3, about to execute lines 1 and 2, about to execute lines 1 and 1

Question 4

Partially correct | Mark 2.50 out of 10.00

Consider this class's rep:

```
public class Building {
    private final String buildingName;
    private int numberOfFloors;
    private final int[] occupancyPerFloor;
    private final List<String> companyNames = Collections.synchronizedList(new ArrayList<>());
    private final Set<String> roomNumbers = new HashSet<>();
    private final Set<String> floorplan = Collections.synchronizedSet(roomNumbers);
    ...
}
```

Which of these variables refer to a value of a threadsafe data type?

Select one or more:

- [x] i. buildingName
- [x] ii. numberOfFloors
- [x] iii. occupancyPerFloor
- [ ] iv. companyNames
- [ ] v. roomNumbers
- [ ] vi. floorplan

Your answer is partially correct.

`buildingName` has type `String`, and `numberOfFloors` has type `int`. Both types are immutable, so the values of the types are threadsafe.

`occupancyPerFloor` has type `int[]`, which is mutable and not threadsafe.

`companyNames` has type `List<String>`, which is not automatically threadsafe, but the implementation of `List<String>` used here is a synchronized list wrapper, and `companyNames` is final so it can never be assigned to a different `List`, so this type is threadsafe.

Similarly, the actual type of the `roomNumbers` value is `HashSet`, which is not threadsafe.

But the synchronized set wrapper is threadsafe, so `floorplan` points to a value of a threadsafe data type.

You have correctly selected 2.
The correct answers are: buildingName, numberOfFloors, companyNames,
floorplan

Question 5

Correct | Mark 10.00 out of 10.00

Consider this class's rep:

```
public class Building {
    private final String buildingName;
    private int numberOfFloors;
    private final int[] occupancyPerFloor;
    private final List<String> companyNames = Collections.synchronizedList(new ArrayList<>());
    private final Set<String> roomNumbers = new HashSet<>();
    private final Set<String> floorplan = Collections.synchronizedSet(roomNumbers);
    ...
}
```

Which of these variables are safe for use by multiple threads?

Select one or more:

- [x] i. buildingName
- [ ] ii. numberOfFloors
- [ ] iii. occupancyPerFloor
- [x] iv. companyNames
- [ ] v. roomNumbers
- [ ] vi. floorplan

Your answer is correct.

Not only does the variable's type have to be threadsafe, but the variable itself should be unreassignable. `buildingName` and `companyNames` satisfy that, but reads and writes of `numberOfFloors` may have race conditions.

When using a synchronized collection wrapper, you have to be sure not to keep any aliases to the underlying collection. So `companyNames` is safe because no other variables hold a reference to the underlying `ArrayList`, but `floorplan` is not safe because `roomNumbers` points to the same `HashSet`.

The correct answers are: buildingName, companyNames

Question 6

Incorrect | Mark 0.00 out of 10.00

Consider this class's rep:

```
public class Building {
    private final String buildingName;
    private int numberOfFloors;
    private final int[] occupancyPerFloor;
    private final List<String> companyNames = Collections.synchronizedList(new ArrayList<>());
    private final Set<String> roomNumbers = new HashSet<>();
    private final Set<String> floorplan = Collections.synchronizedSet(roomNumbers);
    ...
}
```

Which of these variables **cannot** be involved in any race condition?

Select one or more:

- [ ] i. buildingName
- [ ] ii. numberOfFloors
- [x] iii. occupancyPerFloor
- [x] iv. companyNames
- [ ] v. roomNumbers
- [ ] vi. floorplan

Your answer is incorrect.

`buildingName` is unreassignable and immutable, so it can't be involved in any race condition.

`companyNames` might still be involved in a race condition caused by (otherwise safe) mutations to the list, e.g.:

```
if (companyNames.size() > 0) { String firstCompany = companyNames.get(0); }
```

If another thread could empty the `companyNames` list between the size check and the `get` call, then this code will fail.

The correct answer is: buildingName

Question 7

Incorrect | Mark 0.00 out of 10.00

If thread B tries to acquire a lock currently held by thread A:

What happens to thread A?

Select one:

- a. blocks until B acquires the lock
- b. blocks until B releases the lock
- c. throws an exception
- d. nothing

Your answer is incorrect.

The correct answer is: nothing

Question 8

Incorrect | Mark 0.00 out of 10.00

If thread B tries to acquire a lock currently held by thread A:

What happens to thread B?

Select one:

- a. blocks until A acquires the lock
- b. blocks until A releases the lock
- c. throws an exception
- d. nothing

Your answer is incorrect.

The correct answer is: blocks until A releases the lock

Question 9

Partially correct | Mark 2.50 out of 10.00

Suppose `list` is an instance of `ArrayList<String>`.

What is true while thread A is in a `synchronized (list) { ... }` block?

Select one or more:

- i. it owns the lock on `list`

i. it owns the lock on `list`

ii. it does not own the lock on `list`

iii. no other thread can use observers of `list`

iv. no other thread can use mutators of `list`

v. no other thread can acquire the lock on `list`

vi. no other thread can acquire locks on elements in `list`

Your answer is partially correct.

You have correctly selected 1.

The correct answers are: it owns the lock on `list`, no other thread can acquire the lock on `list`

Question 10

Incorrect Mark 0.00 out of 10.00

Suppose we run this code:

```
synchronized (obj) {
    // ...
    synchronized (obj) { // 1
        // ...
    }
    // 2
}
```

On the line `// 1` do we experience deadlock?

If we don't deadlock, on the line `// 2`, does the thread own the lock on obj?

Select one:

a. No, we do not experience deadlock; Yes, the thread owns the lock on obj

b. Yes, we experience deadlock; No, the thread does not own the lock on obj

c. Yes, we experience deadlock; Yes, the thread owns the lock on obj

d. No, we do not experience deadlock; No, the thread does not own the lock on obj

Your answer is incorrect.

In Java, a thread is allowed to re-acquire a lock it already owns. The technical term for this is *reentrant locks*.

Acquire and release come in pairs, and `synchronized` blocks on the same object can be safely nested inside each other. This means that a lock actually stores a *counter* of the number of times that its owner has acquired it without yet releasing it. The thread continues to own it until each acquire has had its corresponding release, and the counter has fallen to zero. So on the "do we own the lock" line, the thread does still have a lock on `obj`.

Nested synchronization on the same lock happens frequently, e.g. if a `synchronized` method is recursive, or if one `synchronized` method calls another `synchronized` method on `this`.

The correct answer is: No, we do not experience deadlock; Yes, the thread owns the lock on obj

Question 11

Correct Mark 10.00 out of 10.00

In the code below three threads 1, 2, and 3 are trying to acquire locks on objects `alpha`, `beta`, and `gamma`

In the code below three threads 1, 2, and 3 are trying to acquire locks on objects `alpha`, `beta`, and `gamma`.

Thread 1

```
synchronized (alpha) {
    // using alpha
    // ...
}

synchronized (gamma) {
    synchronized (beta) {
        // using beta & gamma
        // ...
    }
}
// finished
```

Thread 2

```
synchronized (gamma) {
    synchronized (alpha) {
        synchronized (beta) {
            // using alpha, beta, & gamma
            // ...
        }
    }
}
// finished
```

Thread 3

```
synchronized (gamma) {
    synchronized (alpha) {
        // using alpha & gamma
        // ...
    }
}

synchronized (beta) {
    synchronized (gamma) {
        // using beta & gamma
        // ...
    }
}
// finished
```

This system is susceptible to deadlock.

For each of the scenarios below, determine whether the system is in deadlock if the threads are currently on the indicated lines of code.

Scenario A

Thread 1 inside `using alpha`
Thread 2 blocked on `synchronized (alpha)`
Thread 3 finished

Scenario B

Thread 1 finished
Thread 2 blocked on `synchronized (beta)`
Thread 3 blocked on 2nd `synchronized (gamma)`

Select one:

- a. A not deadlock, B deadlock
- b. A deadlock, B deadlock
- c. A deadlock, B not deadlock
- d. A not deadlock, B not deadlock

Your answer is correct.

Scenario A : Thread 1 will exit the top synchronized block, release the lock on `alpha`, and the system will continue.

Scenario B : Thread 2 has acquired the lock on `gamma` and is awaiting `beta`. Thread 3 has `beta` and wants `gamma`. Deadlock.

The correct answer is: A not deadlock, B deadlock

Question 12

Incorrect | Mark 0.00 out of 10.00

In the code below three threads 1, 2, and 3 are trying to acquire locks on objects `alpha`, `beta`, and `gamma`.

Thread 1

```
synchronized (alpha) {
    // using alpha
    // ...
}

synchronized (gamma) {
    synchronized (beta) {
        // using beta & gamma
        // ...
    }
}
// finished
```

Thread 2

```
synchronized (gamma) {
    synchronized (alpha) {
        synchronized (beta) {
            // using alpha, beta, & gamma
            // ...
        }
    }
}
// finished
```

Thread 3

```
synchronized (gamma) {
    synchronized (alpha) {
        // using alpha & gamma
        // ...
    }
}

synchronized (beta) {
    synchronized (gamma) {
        // using beta & gamma
        // ...
    }
}
// finished
```

This system is susceptible to deadlock.

For each of the scenarios below, determine whether the system is in deadlock if the threads are currently on the indicated lines of code.

Scenario C

Thread 1 running `synchronized (beta)`
Thread 2 blocked on `synchronized (gamma)`
Thread 3 blocked on 1st `synchronized (gamma)`

Scenario D

Thread 1 blocked on `synchronized (beta)`
Thread 2 finished
Thread 3 blocked on 2nd `synchronized (gamma)`

Select one:

- a. C not deadlock, D deadlock
- b. C deadlock, D deadlock
- c. C deadlock, D not deadlock
- d. C not deadlock, D not deadlock

Your answer is incorrect.

Scenario C : Thread 1 can successfully acquire the lock on `beta`, then exit the synchronized block, and one of the other threads will be able to acquire the lock on `gamma`. (As we saw in scenario B, they could deadlock later!)

Scenario D : Thread 1 has acquired the lock on `gamma` and is awaiting `beta`. Thread 3 has `beta` and wants `gamma`. Deadlock.

The correct answer is: C not deadlock, D deadlock

Question 13

Incorrect | Mark 0.00 out of 10.00

In the code below three threads 1, 2, and 3 are trying to acquire locks on objects `alpha`, `beta`, and `gamma`.

Thread 1

```
synchronized (alpha) {
```

Thread 2

```
synchronized (gamma) {
```

Thread 3

```
synchronized (gamma) {
```

```
    // using alpha
    // ...
}

synchronized (gamma) {
    synchronized (beta) {
        // using beta & gamma
        // ...
    }
}
// finished
```

```
    synchronized (alpha) {
        synchronized (beta) {
            // using alpha, beta, & gamma
            // ...
        }
    }
}
// finished
```

```
    synchronized (alpha) {
        // using alpha & gamma
        // ...
    }
}

synchronized (beta) {
    synchronized (gamma) {
        // using beta & gamma
        // ...
    }
}
// finished
```

This system is susceptible to deadlock.

In the previous problem, we saw deadlocks involving `beta` and `gamma`.

What about `alpha`?

Select one:

- a. there are no deadlocks involving `alpha`
- b. there is a possible deadlock where thread 1 owns the lock on `alpha`
- c. there is a possible deadlock where thread 2 owns the lock on `alpha`
- d. there is a possible deadlock where thread 3 owns the lock on `alpha`

Your answer is incorrect.

We can reason about it this way: in order to encounter deadlock, threads must try to acquire locks in different orders, creating a cycle in the graph of who-is-waiting-for-who.

So we look at `alpha` vs. `beta`: are there two threads that try to acquire these locks in the opposite order? No. Only thread 2 acquires them both at the same time.

Next we look at `alpha` vs. `gamma`: are there two threads that try to acquire these locks in the opposite order? No. Both thread 2 and thread 3 acquire both locks, but both of them acquire `gamma` first, then `alpha`.

The correct answer is: there are no deadlocks involving `alpha`

Finish review

Lab 13 Recording Jump to... Lab Exercise 13.1 ARBinHeap C