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;

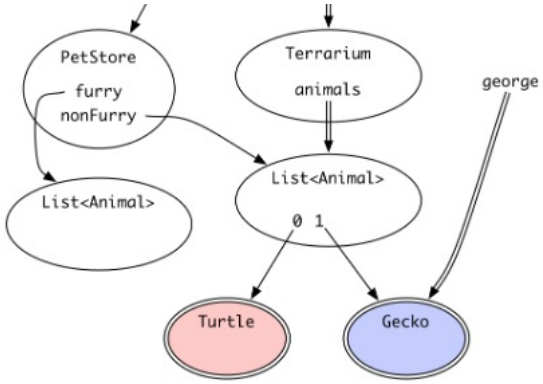
Your answer is correct.

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

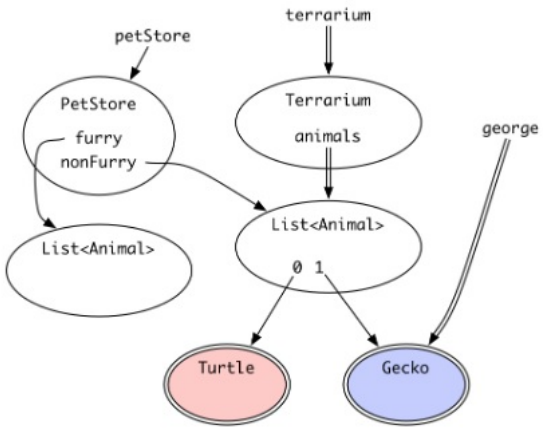
Your answer is incorrect.

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

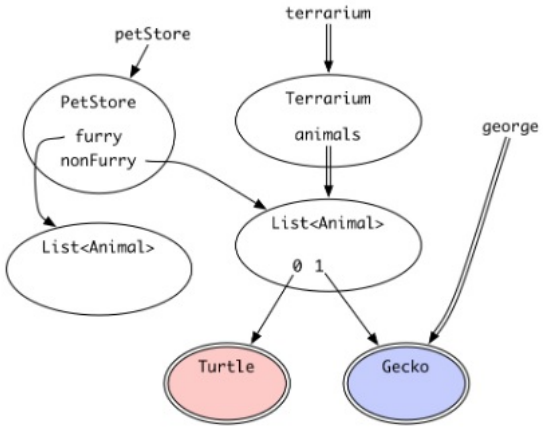
Your answer is incorrect.

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