TEST

Software Systems: Programming

course code: 202001024
date: 15 January 2021
time: 9:00 – 12:00

SOLUTIONS

General

- You may use the following (unmarked) materials when making this test:
  - Module manual.
  - Slides of the Programming topics.
  - The book
    David J. Eck. *Introduction to Programming Using Java*. Version 8.1.2, December 2020.
  - A dictionary of your choice.
  - IntelliJ and/or Eclipse.
  - Documentation of Java 11
    `https://docs.oracle.com/en/java/javase/11/docs/api/`

  You are *not* allowed to use any of the following:

  - Solutions of any exercises published on Canvas (such as recommended exercises or old tests);
  - Your own materials (copies of (your) code, solutions of lab assignments, notes of any kind, etc.).
- When you are asked to write Java code, follow code conventions where they are applicable. Failure to do so may result in point deductions. It is recommended that you use IntelliJ and/or Eclipse to write code.
- You do *not* have to add Javadoc or comments, unless explicitly asked to do so. Invariants, preconditions and postconditions should be given only when they are explicitly asked.
- You are not allowed to leave the room during the first 30 minutes or the last 15 minutes of the exam.
- Place your student ID card on the table as well as documentation that grants extra time (if applicable).

Question 1 (4 points)

a. *(2 pts)* Explain the general difference between compile-time errors and runtime errors.

b. *(1 pts)* Give an example of a compile-time error

c. *(1 pts)* Give an example of a runtime error

Answer to question 1

Compile-time errors occur when compiling Java code to a Java bytecode file, such as problems with static typing or syntax errors. An example would be calling methods that do not exist, syntax errors like missing semicolons or bad braces, etc. Runtime errors occur when running a program, such as array index out of bounds, null pointer exceptions, division through zero, type cast errors, etc.

a. +2 for correct answer
b. +1 for a good example
c. +1 for a good example

Question 2 (4 points)

a. *(2 pts)*

What is printed if we compile and run the following code fragment:

```
int a = 1;
int b = 1;
System.out.println(a + "+" + b + "=" = a + b);
```

(a) `1+1=11`
(b) `1+1=1+1`
(c) There will be a compilation error
(d) There will be a runtime error

Explain your answer.

b. *(2 pts)* The intended result was `1+1=2`. Explain how to change the code to get this result, without explicitly using the numbers 1 and 2.

Answer to question 2

The correct answer is that there will be a compilation error because `=` should be `+`. Easy points if you copied the code to Eclipse or IntelliJ and understood what was wrong. To fix the code to get `1+1=2` two things need to be changed: `=` should be changed to `+` and parentheses need to be added to get `System.out.println(a + "+" + b + "=" + (a + b));`. Variations of this are also correct, for example introducing an extra variable `int c = a + b;` although unnecessary.

a. +2 for correct explanation
b. +1 for changing `=` to `+` and +1 for adding parentheses

Question 3 (4 points)

Consider the following Java classes:

```
public class A {
    protected int value = 12;
    protected A() {
    }
    public A(int value) {
        value = value;
    }
    protected int getValue() {
        return value;
    }
}

public class B extends A {
    private B() {
        value = 9;
    }
    public B(int value) {
        super(value);
    }
    public int getValue() {
        return super.getValue() + 1;
    }
}
```

Consider also the following Java code:

```
B b = new B(10);
System.out.println("Value: " + b.getValue());
```

What is printed when compiling and executing the given code fragment? Explain your answer.

a. `Value: 9`
b. `Value: 10`
c. `Value: 11`
d. `Value: 12`
e. `Value: 13`
f. Nothing, there will be an error.

Answer to question 3

The correct answer is `Value: 13`.

- +1 for noticing that `B(int)` calls the superconstructor `A(int)`
- +1 for noticing that `A(int)` does not do anything (because variable shadowing)
- +1 for noticing that the value is initially set to 12 by the initializer of `A`
- +1 for noticing that the `getValue` implementation of `B` is called, which executes its super and adds 1, returning 13.

Question 4 (4 points)

Consider the following Java classes:

```
public interface A {
    public int a();
}
public class ExampleClass extends A {
    private ExampleClass() {
```

```
    }
    public int a() {
        return b();
    }
    private static int b() {
        return c();
    }
    protected final int c() {
        return 4;
    }
    public final void main(String[] args) {
        ExampleClass.a();
        A ex = new ExampleClass();
        ex.a();
    }
}
```

This code fragment has three compile-time and/or runtime errors. Explain these errors by referring to what is wrong, why that is wrong, and how it could be fixed.

Answer to question 4

Only the first three mentioned errors count. If the answer includes more, only grade the first three.

- (+1) **extends** should be **implements** because `A` is an interface. Simply substitute.
- (+1.5) **static** method `b()` cannot call **non-static** method `c()` so either `b()` should not be **`static`** or `c()` should become **`static`**.
- (+1.5) Cannot invoke `Exampleclass.a()` since it is not static. Either remove the line or make `a` and `c` static (also in the interface A)
- (+0.5) main should be **`static`** and not **`final`**. This would be true, except that it doesn't give a compile-time or runtime error. Java will simply not recognize the function main as a valid entry-point. It is allowed to define and use a method **`public final void`** `main(String[] args)`. So this does not count for full points.
- Wrong: "methods cannot be **`final`**". Actually, methods can be **`final`** meaning subclasses may not override them.

Question 5 (4 points)

Model-View-Controller is a well known design pattern. Imagine we program a tic tac toe game. Explain for the following examples whether they belong to the view, to the controller, or to the model, and why.

a. *(1 pt.)* A ButtonEventListener that receives the event that someone clicked one of the empty spaces on the game board.
b. *(1 pt.)* A TicTacToeCanvas that displays the current state of the game.
c. *(1 pt.)* A TicTacToeBoard that stores the current game board.
d. *(1 pt.)* An algorithm that checks whether someone won the game.

Answer to question 5

In principle:

- View are classes that display things to a user, like representations of the data in the model
- Controller are classes that handle user input and decide what to do with it
- Model are basically all classes that are not View or Controller, such as the data, the logic, algorithms, etc.

As a guideline, things you would replace if you plug in a different view and/or a different controller are part of the view and/or the controller. Algorithms are usually part of the model, unless they are very specifically part of viewing or controlling.

Only model/view/controller is insufficent. If the answer has all four correct without an explanation, award 1 point for the entire question, but otherwise no points for only the word "model" or "view"'or "controller". The questions clearly asks to **explain**.

a. (+1) Controller: reacts to user input
b. (+1) View: displays the board to the user
c. (+1) Model: stores the data of the state of the board; not part of displaying or reacting to user input
d. (+1) Model: it's not related to displaying/reacting; also award points for: "data or logic is part of the model". Controller/View is wrong, because if we would replace the controller/view, we wouldn't want to replace this algorithm.

Question 6 (15 points)

Suppose we have a class hierarchy with the class `Appliance`, which has subclasses `Table` and `Chair` and `Bookcase`. The `Bookcase` class has methods `addBook(Book book)`, `removeBook(Book book)`. The `Table` class has methods `addItem(Item item)` and `removeItem(Item item)`. Both `Appliance` and `Book` implement the interface `Item`.

Consider we have the following code:

```
Table t = new Table();
Appliance b = new Bookcase();
Book book = new Book();
```

a. *(6 pts)* For each of the following lines, determine whether or not they give a runtime error and/or a compilation error and give your explanation.
   (a) `t.addBook(book);`
   (b) `Bookcase bc = b;`
   (c) `((Table) b).addItem(book);`

b. *(6 pts)* Imagine we now want chairs to have a color. We give three methods to accomplish this. For each of these methods, explain when it would be appropriate to use that method.
   (a) Add a `color` field to the `Appliance` class and appropriate getters and setters.
   (b) Add a `color` field to the `Chair` class and appropriate getters and setters.
   (c) Extend `Chair` by a `ColoredChair` subclass which has a `color` field and appropriate getters and setters.

c. *(3 pts)* Your colleague has implemented colored chairs using (empty) interfaces `Red`, `Blue`, etc, and subclasses **`class`** `RedChair` **`extends`** `Chair` **`implements`** `Red`, etc. Your colleague explains that you can check the color of a chair using the **`instanceof`** keyword. There is however an important difference between this solution and the other solution. What can you do with the solutions of subquestion b that you cannot do with this solution?

Answer to question 6

a. The traditional static/dynamic typing question. Both the correct error and the correct explanation are required.

(a) (+2) compilation error, because (static) type `Table` does not have a method with signature `addBook(Book)`.
(b) (+2) compilation error, because (static) type `Appliance` cannot be assigned to a variable of type `Bookcase` since `Appliance` is not a subtype of `Bookcase`.
(c) (+2) runtime error, because (dynamic) type `Bookcase` cannot be typecast to type `Table`. Note: There is no compilation error, because the type `Table` has a method with signature `addItem(Item)` and `book` is of type `Book` which is a subtype of `Item`.

b. The question was about in which case would you make which choice?

(a) (+2) When we want all appliances to have color
(b) (+2) When we want all chairs to have color
(c) (+2) When we want some chairs to have color and some chairs to not have color

c. (+3) You cannot change the color of a chair, since you cannot change the type of an object. If you have a field with the color, then you can change the color of the chair.

Question 7 (10 points)

A mathematical number sequence to rationally approximate $\sqrt{2}$ is the Pell sequence:
- $P_0 = 1$
- $P_1 = 2$
- $P_n = 2P_{n-1} + P_{n-2}$

The first Pell numbers are 1, 2, 5, 12, 29, 70, 169, ... The fraction $\frac{P_n + P_{n+1}}{P_{n+1}}$ approximates $\sqrt{2}$.

a. *(5 pts)* Implement a method **`long`** `pellRecursive(`**`int`** `n)` that computes and returns the Pell number n. This method must be **recursive**.

b. *(5 pts)* Implement a nonrecursive method **`long`** `pellArray(`**`int`** `n)` that computes and returns the Pell number n. This method **must not** call any methods. Instead, you must use an array of **`long`**s to compute the Pell numbers, up to and including number n, then return Pell number n.

Answer to question 7

a. An example solution:

```
public static long pellRecursive(int n) {
    if (n == 0) return 1;
    if (n == 1) return 2;
    return 2*pellRecursive(n-1) + pellRecursive(n-2);
}
```

- (+1) Correct method signature and return type. (don't care about static/public/etc)
- (+1) Correctly return 1 or 2 if n equals 0 or 1
- (+2) Correctly call the recursive function for n-1 and n-2
- (+1) Correctly computes the result based on the recursive results

b. An example solution:

```
public static long pellArray(int n) {
    if (n == 0) return 1;
    if (n == 1) return 2;
```

```
        long[] results = new long[n+1];
        results[0] = 1;
        results[1] = 2;
        for (int i=2; i<=n; i++) {
            results[i] = 2*results[i-1] + results[i-2];
        }
        return results[n];
    }
```

- (+1) Correct method signature and return type and no method calls. (don't care about static/public/etc)
- (+1) Create a `long` array of appropriate size (usually `n+1`).
- (+1) Give the correct result for n=0 and n=1. (either using the array or directly returning)
- (+1) Use an appropriate loop (such as for, while)
- (+1) Correctly computes each next Pell number and store in the array, returning the correct result

Question 8 (25 points)

In this question, you are going to program a class `TetrisCompetition` that collects the results of a Tetris competition. In the competition, players try to get as many points as they can playing Tetris. Of all their attempts, we use the **median** score to determine who wins the competition.

The median score is any of the values such that at most half of the population is less than the proposed median and at most half is greater than the proposed median. For example, the median of $\{1,3,3,6,7,8,9\}$ is 6. When the number of values is *even*, the median is the average of the middle two numbers. For example, the median of $\{1,2,3,4,5,6,8,9\}$ is 4.5.

a. *(4 pts)* Create and implement a Java class `TetrisCompetition` with a method `int median(List<Integer> numbers)` that computes the median number of a given list of numbers. Hint: you can use Java's `Collections.sort` method.

b. *(4 pts)* A `TetrisCompetition` object has to keep track of all obtained scores in the competition. Add an appropriate field or appropriate fields to your `TetrisCompetition` class and implement a method `addResult(String name, int score)` which adds one score to the scores recorded by the `TetrisCompetition` object.

c. *(4 pts)* Implement a method `String getWinner()` which returns the winner of the competition.

d. *(2 pts)* Explain your design decisions regarding how you store the participants and their scores.

e. *(2 pts)* Explain your design decisions regarding how you compute and return the winner of the competition.

f. *(2 pts)* Explain your design decisions of using **`public`**, **`protected`** and **`private`** for the fields and methods of your class.

g. *(2 pt)* Explain your design decisions regarding how you use initializers and/or constructors to initialize objects of your class.

h. *(5 pts)* Define appropriate preconditions and postconditions for `addResult`. Pay attention that your postcondition should describe the change to the state of the `TetrisCompetition` object after a call to `addResult`.

Answer to question 8

a.
```
public class TetrisCompetition {
    private static int median(List<Integer> numbers) {
        Collections.sort(numbers);
        int size = numbers.size();
        if (size % 2 == 0) {
            return (numbers.get(size/2-1)+numbers.get(size/2))/2;
        } else {
            return numbers.get(size/2);
        }
    }
}
```

- (+1) correct definition of method with specified method signature (can be static or not, that doesn't matter here)
- (+1) method median somehow sorts the numbers, e.g. `Collections.sort(numbers)` or `numbers.sort(null)`)
- (+1) Correct result for odd number of numbers (middle one)
- (+1) Correct result of even number of numbers (average of two middle numbers)

b.
```
public class TetrisCompetition {
    private Map<String, List<Integer>> scores = new HashMap<>();

    public void addResult(String name, int score) {
        scores.putIfAbsent(name, new ArrayList<>());
        scores.get(name).add(score);
    }
}
```

- (+1) a field `Map<String, List<Integer>>` or something very similar.
- (+1) correct initializing of the field, either directly, or in an initializer or constructor, e.g. to `HashMap<>()`.
- (+1) `addResult` should correctly add a fresh `List` implementation if this is the first time we see `name`.
  Also correct:
  `if (!scores.contains(name)) scores.put(name, new ArrayList<>());`
  Or any implementation that correctly handles the case where `name` is new.
- (+1) `addResult` should correctly add the score to the scores of the correct player

Complicated methods with multiple fields in the class are incorrect. Points are still awarded for correctly adding the results of the score, but full points are not awarded, since students should know that a `Map` is most appropriate in this case.

c.
```
public String getWinner() {
    String best = null;
    int bestMedian = 0;
    for (String name : scores.keySet()) {
        int currentMedian = median(scores.get(name));
        if (best == null || bestMedian < currentMedian) {
            best = name;
            bestMedian = currentMedian;
        }
    }
    return best;
}
```

- (+2) some loop over all participants that computes the median of the current participant
- (+2) some method to track the winner so far, and update it if the current participant has a higher median than the winner, and correctly returns the winner

Answers that user Java streams are also correct, but only if they actually return the winner. It is not necessary to "cache" the best median to get full points, but it would be more efficient.

d. (+2) Some valid explanation, e.g., using a map is a natural way to store information about a player, in this case a list of scores, which is best represented as a list (or collection) of integers.

e. (+2) Some valid explanation, e.g., we use a loop over all participants, and we keep track of the best participant so far. (Optionally: we also keep track of the median of that participant, so we don't need to recompute it each time. Optionally: we use Java streams, which lets us get the participant with the highest median on-the-fly).

f. (+2) Some valid explanation; but should be: fields are **private** because of encapsulation (or maaaaybe **protected** if the answer explicitly argues *why* subclasses should be able to access the field directly, but there is no cause for this in the text); any answer where the field with scores is **public** is simply wrong; methods: depends on the explanation; for example `median` could be **private** or **public** (do I want other classes to use it?), but probably `addResult` and `getWinner` should be **public** otherwise you can't use the class (again depends on explanation)

g. (+2) easy points: you need to initialize the field that maintains the scores, whether this happens in the constructor or an initializer doesn't really matter. If they used a constructor, it should be a constructor without parameters that just initializes the field(s). If they used a constructor with a parameter without a clear reason, no points.

h. Examples of good preconditions:

- (+1) `name != null`
- (+2) `name != null && score >= 0` (although scores are not explicitly stated to be non-negative)
- (+2) `name != null` and explicitly remarking that scores can also be negative, showing that the student considered this

Examples of good postconditions:

Award 2 points for informally stating that the score is added to the list of the player in the field scores. (They must at least mention the name of the updated field, e.g. `scores`)

Award 3 points for informally stating that the score is added to the list of the player in the field scores *and that nothing else changes*.

Award 2 points for more formal specifications like:

```
scores.get(name).size() ==
    \old(scores.getOrDefault(name, new ArrayList<Integer>()).size()) + 1

\old(scores.contains(name)) ? scores.get(name).size() == 1 :
      scores.get(name).size() == \old(scores.get(name).size()) + 1}
```

Award 3 points for a specification like:

```
scores.get(name).get(scores.get(name).size()-1) == score &&
scores.get(name).size() ==
    \old(scores.getOrDefault(name, new ArrayList<Integer>()).size()) + 1
```

Or even:

```
    scores.get(name).get(scores.get(name).size()-1) == score &&
        (scores.get(name).size() == 1 ||
            (scores.get(name).subList(0, scores.get(name).size()-1)
                            .equals(\old(scores.get(name)))
            )
        )
```

Question 9 (10 points)

The results from the competition of the previous question are stored in a file, which is a sequence of names and scores, separated by whitespace (one or more spaces, tabs, newlines), for example:

```
linda 1523   jane 145  carl
41 thomas 331 thomas 366 john 321  annet 142 linda 1034
thomas 399 carl 99 thomas 901

anon 911 annet 420 linda 9999
```

a. *(5 pts)* Implement a static method `TetrisCompetition processFile(String filename)` which reads a file (given by the filename) with competition results, and adds the results to a new `TetrisCompetition` object which is returned by the method. You may assume that a given file follows this file format. Any exceptions should be thrown (not caught in `processFile`).

b. *(5 pts)* Implement a `main` method that expects a command line argument with a filename of a file with competition results. Read the file, then print the winner of the competition to standard output. Add appropriate error messages for cases such as files not existing, command line parameters are missing, etc.

Answer to question 9 Example of correct implementation:

```
public static TetrisCompetition processFile(String filename)
        throws FileNotFoundException {
    TetrisCompetition competition = new TetrisCompetition();
    try (Scanner sc = new Scanner(new FileReader(filename))) {
        while (sc.hasNext()) {
            String name = sc.next();
            int score = sc.nextInt();
            competition.addResult(name, score);
        }
    }
    return competition;
}

public static void main(String[] args) {
    if (args.length < 1) {
        System.err.println("Expected␣command␣line␣argument!!");
    } else {
        try {
            TetrisCompetition competition = processFile(args[0]);
            System.out.println("Winner:␣" + competition.getWinner());
        } catch (FileNotFoundException e) {
            System.err.println("The␣file␣" + args[0] + "␣was␣not␣found!");
```

```
        }
    }
}
```

a. - (+1) Correctly make a `FileReader` or `FileInputStream` and close it, for example with a try-with-resources block (the point is that the resource should be closed afterwards!)
- (+1) Use a `Scanner` that wraps the `FileReader` or `FileInputStream` (either directly or via a `BufferedReader`). Something like **`new`** `Scanner(filename)` is incorrect, since that will scan the String instead of the file.
- (+1) correctly obtain each name and score (simple if you use a `Scanner`)
- (+1) correctly create, update, return the `TetrisCompetition` object
- (+1) correctly handle the `FileNotFoundException` (either catch-and-rethrow, or simply don't catch it)

Solutions not using a `Scanner` can still get some points, but not full points.

b. - (+1) Correct method signature for `main`
- (+1) Use `args[0]` for the file name and give an error if `args.length == 0`
- (+1) The call to `processFile` is correct (use returned `TetrisCompetition` in next step)
- (+1) Catch the exception(s) of `processFile` and report an error
- (+1) Print the winner of the competition to `System.out`.

It doesn't matter if errors go to `System.out` or `System.err`.

Question 10 (10 points)

Consider the following Java class.

```
1  public class ProducerConsumerExample {
2      public static class StackOfNumbers {
3          private final StackOfNumbers rest;
4          private final int number;
5
6          StackOfNumbers(StackOfNumbers rest, int number) {
7              this.rest = rest;
8              this.number = number;
9          }
10
11         public StackOfNumbers getRest() {
12             return rest;
13         }
14
15         public int getNumber() {
16             return number;
17         }
18     }
19
20     private static volatile StackOfNumbers top = null;
21
22     public static class Producer implements Runnable {
23         private int from, to;
24
25         public Producer(int from, int to) {
26             this.from = from;
```

```
27              this.to = to;
28          }
29
30          @Override
31          public void run() {
32              for (int i = from; i < to; i++) {
33                  top = new StackOfNumbers(top, i);
34              }
35          }
36      }
37
38      public static class Consumer implements Runnable {
39          private int count;
40
41          public Consumer(int count) {
42              this.count = count;
43          }
44
45          @Override
46          public void run() {
47              for (int i = 0; i < count; i++) {
48                  while (top == null) continue; // wait until we have something
49                  System.out.println("next item: " + top.getNumber());
50                  top = top.getRest();
51              }
52          }
53      }
54
55      public static void main(String[] args) {
56          Thread prod1 = new Thread(new Producer(0, 10));
57          Thread prod2 = new Thread(new Producer(10, 20));
58          Thread prod3 = new Thread(new Producer(20, 30));
59          Thread cons = new Thread(new Consumer(30));
60
61          prod1.start();
62          prod2.start();
63          prod3.start();
64          cons.start();
65          try {
66              prod1.join();
67              prod2.join();
68              prod3.join();
69              cons.join();
70          } catch (InterruptedException ignored) {
71          }
72      }
73  }
```

The **volatile** keyword (see the book by Eck, 12.1.4) indicates to Java that a variable or field might be changed by other threads and forbids Java from storing a "local copy". Without this keyword, the **while** loop of line 48 will loop forever, even if another thread changes the value of `top`.

a. *(3 pts.)* Someone runs the above `main` method and sees the numbers 0 to 29 in a seemingly random order. However the numbers are not completely random: there are constraints on the order that numbers appear. Describe these constraints.

b. *(3 pts.)* The next time they run the `main` method, they find that the program does not terminate. Explain how this is possible.

c. *(4 pts.)* Make the code thread-safe, ensuring that it always terminates and does not crash.

Answer to question 10

a. 2 points for: the result is an interleaving of the three lists $\{9,8,7,6,5,4,3,2,1,0\}$, $\{19,18,17,16,15,14,13,12,11,10\}$, $\{29,28,27,26,25,24,23,22,21,20\}$ with some kind of explanation about using a stack resulting in this. Careful: interleaving means that threads can interrupt each other, resulting in possible interleavings like $\{9,8,7,19,18,29,28,27,26,6,5,4,...\}$. If the answer is that threads wait for each other then that is not correct and results in deduction of 1 point.

3 points for: numbers are pushed to the stack in order, but the consumer can already consume some numbers while pushing. So if pushing 0-9, if the consumer consumes 3, then it will encounter 2,1,0 in that order (maybe interleaved) later. If the consumer consumes 3, then 6, then 9, then it will see 8,7,5,4,2,1,0 later, possibly interleaved with the other two lists.

b. (+3) There is a race condition when two threads update top at the same time. This can happen when either two (or more) producers simultaneously update `top`, or when one (or more) producer(s) and the consumer update `top`. As a result, sometimes a number is never put on the stack. The consumer waits until it has seen 30 numbers, so if any number is missing, the consumer will not finish.

Answers like 'it gets stuck in the while loop' are insufficient, because they don't explain why this happens. Answers about **`volatile`** or 'the ignored exception' are also wrong. Removing **`volatile`** would guarantee a deadlock actually.

c. - (+2) some kind of synchronization that removes the race condition between producers, for example a shared lock, or synchronizing on a shared object. 0 points for synchronizing on the `top` object, it changes all the time. Award only 1 point if the entire **`for`**-loop is inside the synchronization.

   Just adding **`synchronized`** to methods is wrong because this synchronizes on the different `Runnable` subclasses, instead of on a shared object.

   - (+2) some kind of synchronization to remove the race condition between producers and consumers: a shared lock, synchronize on a shared object, maybe even using `wait` and `notify`. It is fine if the **`while`** loop is outside the critical section! Do not award points if the answer puts the entire **`while`** loop inside the critical section without using wait/notify, because this results in a deadlock if the consumer enters when `top==`**`null`**. If the stack is empty, then the consumer must leave the critical section if it entered it. `getResult` and the update to `top` must occur inside the same critical section, otherwise it is possible to see the same number twice and some numbers never!

   Example:

```
// new field
private final static Object sync = new Object();

// in producer run()
for (int i = from; i < to; i++) {
    synchronized (sync) {
        top = new StackOfNumbers(top, i);
    }
}

// in consumer run()
for (int i = 0; i < count; i++) {
    while (top == null) continue; // wait until we have something
```

```
    synchronized (sync) {
        System.out.println("next␣item:␣" + top.getNumber());
        top = top.getRest();
    }
}
```

Alternative:

```
// new field
private final static Lock lock = new ReentrantLock();

// in producer run()
for (int i = from; i < to; i++) {
    lock.lock();
    top = new StackOfNumbers(top, i);
    lock.unlock();
}

// in consumer run()
for (int i = 0; i < count; i++) {
    while (top == null) continue; // wait until we have something
    lock.lock();
    System.out.println("next␣item:␣" + top.getNumber());
    top = top.getRest();
    lock.unlock();
}
```

Alternative using wait/notify:

```
// new field
private final static Object sync = new Object();

// in producer run()
for (int i = from; i < to; i++) {
    synchronized (sync) {
        top = new StackOfNumbers(top, i);
        sync.notify();
    }
}

// in consumer run()
for (int i = 0; i < count; i++) {
    synchronized (sync) {
        while (top == null) {
            try {
                sync.wait();
            } catch (InterruptedException ignored) {
            }
        }
        System.out.println("next␣item:␣" + top.getNumber());
        top = top.getRest();
    }
}
```

Question 11 (3 points)

Consider the following headline: “Australian tech unicorn Canva suffers security breach - Hacker claims to have stolen the data of 139 million Canva users.“

a. Which of the three important security properties was violated here?

b. What are the other two security properties?

Answer to question 11

a. (+1) Confidentiality

No points for mentioning all 3.

b. (+1) Integrity (+1) Availability

Question 12 (2 points)

What is multi-factor authentication and what advantage does it have? Give an example of multi-factor authentication.

Answer to question 12

(+1) The commonly used password has a number of downsides (password reuse, easy to phish, hard to manage). With multi-factor authentication an additional ”factor” is used to to authenticate the user. (+1) Such additional factor is for example ”something the user has”. A time-based ”one-time-password” is an example of such an authentication factor. (typical factors: something the user knows, has, is)

Question 13 (5 points)

Next to a 3-digit (0-9) combination lock you find a piece of paper containing the following text:

```
5d8f6cce532a7aeb57196be62344095936793400b3aeb3580d248b17d5518a86: 338
fc71f2d6d38dbfc752ecaf2262916dc8ad99a34243d47b34691f9f8a3afaeffd:
```

You suspect that these are hexadecimal representations of the output of the application of the SHA-256 hash to a string representing the code for the lock. That is, the SHA-256 hash of the string `"338"` is `"5d8f6c.."`. You try 338, but the lock does not open. That was probably the old combination. Write a small program that tries to find which 3-digit combination matches the digest ”fc71f2...”. What is the current combination of the lock?

To get you started, we provide you with the following two snippets of code:

```
public static String hex(byte[] bytes) {
    StringBuilder result = new StringBuilder();
    for (byte aByte : bytes) {
         result.append(String.format("%02x", aByte));
    }
    return result.toString();
}

MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] hash = md.digest(text.getBytes(StandardCharsets.UTF_8));
```

In your answer, give both the current combination of the lock as well as the source code of your solution.

Answer to question 13

- (+2) Correct answer: 623
- (+3) Code that tries all 1000 possibilities in a loop and checks whether one matches the given SHA-256 hash. Only give 2 points for code that does not correctly test ”000”, ”001”, etc.

Example solution:

```
public static void main(String[] args) {
    String target =
        "fc71f2d6d38dbfc752ecaf2262916dc8ad99a34243d47b34691f9f8a3afaeffd";
    try {
        for (int i=0; i<1000; i++) {
            String pass = String.format("%03d", i);
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(pass.getBytes(StandardCharsets.UTF_8));
            if (hex(hash).equals(target)) {
                System.out.println(i);
            }
        }
    } catch (NoSuchAlgorithmException ignored) {
    }
}
```

(Most student answers probably have a few more lines to go from a number to a three digit String)