TECHNISCHE UNIVERSITEIT EINDHOVEN

Examination 2IP90
20 January 2015, 18:00–21:00

corrected version

This exam consists of 4 questions on 4 pages.

- Put your name, number, and the date of today at the top of *every Java file* you submit.
- Add comments to your code where clarification is needed.
- Before you submit your solutions, check that you have included all the files you want to submit and that you have saved them.
- Submit your solutions as compilable `.java` files in the folder `2IP91submitfolder` on your desktop (non-Windows users choose a name themselves).
- Do not use named packages for your code. Use only letters, digits, and underscores in your file names.
- **Before you leave the exam room, report to a supervisor to verify that your work has been submitted.**
- You are allowed to consult on your laptop the course material (lecture slides, reader, programs you have made during the course) and the Java API.
- Use of the internet or other means of communication is *not* allowed during the examination.

---

1 Miscellaneous *(24 pt)*

Submit your answers to these questions in a file Miscellaneous.java.

For answers that are not Java code but text, such as question 1.1 below, insert the text as comment (between `/*` and `*/`. Submit a compilable Java file.

Precede every answer with the number of the question in comment, e.g.,

```
/*
 * 1.1
 */
```
answer...

(6) 1. True or false? There can be only one method named *foo* in a class.

(6) 2. True or false? There can be only one local variable named *x* in a class.

(6) 3. Give two reasons why the following method may not always execute the body *B* exactly `n` times. `n` is an `int` variable.

```
for (int i=0; i<n; i++) {
     B
}
```

(6) 4. What is the number of objects (instances) of class C created by the following method?

```
void create() {
     int n;
     C aap;
     C noot;
     C[] someCs;
     n = 3;
     someCs = new C[n];
}
```

2 Circular Primes *(28 pt)*

Submit your answers to these questions as a class `CircularPrimes`.

A *prime number*, or *prime* for short, is a number that is only divisible by itself and 1 (not considering negative numbers here). E.g., 7, 43, and 101. But not 27, since 27 is divisible by 3 and 9.

A *circular prime* is a prime for any rotation of its digits: For example:

- 37 is a circular prime, since 37 and 73 are prime.
- 113 is a circular prime, since 113, 131, and 311 are prime.
- 907 is prime but not a circular prime, since 790 is not prime.

For these questions, only integer computations are allowed, in particular, don't use String calculations.

You may assume in all questions that argument `n` is a positive number.

(7) 1. Write a method with header `boolean isPrime(int n)` that returns whether or not `n` is prime.

(7) 2. Write a method with header int largestPowerOfTen(int n), that computes the largest power of 10 less or equal to `n`. E.g., $n = 82$ gives 10, $n = 100$ gives 100, and $n = 2015$ gives 1000.

(7) 3. Write a method with header `boolean isCircularPrime(int n)` that returns whether or not `n` is a circular prime. Use the methods from questions 1 and 2.

(7) 4. Write code in the main method to compute all circular primes greater than 1 and less than 1000.

3 Seven Search *(20 pt)*

Given is the following method

```
boolean has7( int[] a) {
    return has7( a, 0, a.length );
}
```

Write a method `boolean has7(int[] a, int lo, int hi)` that takes an int array `a`, that is **sorted in ascending manner**, and two indices `lo` and `hi` and returns `true` if the value 7 occurs in array `a` in the segment $[lo, hi\rangle$. In other words, there is an index i with $a[i] = 7$ and $lo \leq i < hi$. If there is no 7 in this segment of `a`, it should return `false`.

Hence, `has7(a)` will return true if and only if `a` contains a 7.

Examples Suppose `a` is `[1,2,4,7,8,8,9]`, then

- `has7(a, 0, 7)` returns `true`,
- `has7(a, 0, 3)` returns `false`,
- `has7(a, 4, 5)` returns `false`.

Your method has to be a *single recursive method*, which does not use loops. All the variables it uses should be local variables. Furthermore, use the following approach.

Suppose `m` $\in [lo, hi\rangle$. If `a[m] < 7` there can be no 7 in the array before index `m`, so `has7(a, lo, m)` is certainly `false` and `has7(a, m, hi)` has the same value as `has7(a, lo, hi)`. Use this to make an efficient search for 7, by each step (roughly) halving the segment of the array you are considering.

4 Humor *(28 pt)*

In the following questions, you are asked to write some classes and add methods and instance variables to these classes. Adhere to the names, types, and method headers that are mentioned. Feel free to add more methods and variables if you think this makes sense. Submit your code either in a single file with the name Humor.java or in one file for each class, named after the class.

An editor of a satirical magazine wants to keep track of his jokes using Java.

(6) 1. Write a class Joke with instance variables for the name of the joke, the text of the joke, the space it takes to print the joke (in mm), and a fun factor (a double between 0.0 and 10.0 that represents how funny the joke is). Add a constructor `Joke(String name, String description, int space, double funFactor)` and a constructor `Joke()` that sets these variables to default values of your choice, and a method `void print()` that prints the data of the joke.

(6) 2. Add a class Humor that contains an ArrayList of the jokes that he wants to use in an issue. Add a method `add(Joke j)` to Humor that adds a joke to the issue.

(4) 3. Add a method `void overview()` to Humor that prints the total amount of space needed to publish all the jokes of the issue.

(6) 4. The humorist introduces a subcategory of *insulting jokes*. An insulting joke has an instance variable of type `String` with the name of the social group that is primarily insulted by the joke. Add a class InsultingJoke that represents insulting jokes. Use inheritance.

(6) 5. After years of experience, the editor has developed a calculation method to determine the funniness of a joke. It works as follows:

- The funniness of a normal joke (i.e., not insulting) is the fun factor of the joke divided by the space in mm.
- The funniness of an insulting joke is the fun factor multiplied by 1.5 and divided by the space (but see below).
- If it is an insulting joke and the social group is Limburgians, the fun factor is multiplied by 2.0 and divided by the space.

Add methods `double calculateFunniness()` that calculate the funniness of a joke. Extend the method `overview` of question 3 and make it also print the average funniness of the jokes in the issue. (If you answer this question, you don't have to provide a separate answer to question 3.)

For example, piece of code that uses these classes could look like the code below. You could include this in a main or demo method and use it for testing, but this is not required.

```
Issue issue = new Issue();
Joke j;

j = new Joke("Palindrome",
        "A nerdy guy holds a black sign with the text 'Madam, I'm Adam'", 30, 25.0);
issue.add(j);

Joke k;
k = new InsultingJoke("Pastoorke",
        "Guus and Theo meet in Stratums Eind...", 120, 10.0, "Brabanders");
issue.add(k);
```

Good luck!

Grading: The total number of points achieved is the grade g for this examination. The final grade is the result of the following formula rounded to the nearest integer number: $0.6 \cdot g + 0.4 \cdot h$. Here g is the grade for this exam and h is the average of the 6 highest grades for the homework assignments (an assignment that was not submitted is graded with a 0). Furthermore, the grade g has to be at least 5.0 to pass.