TEST

Software Systems: Programming

course code: 201700117
date: 20 January 2020
time: 8:45 – 11:45

General

- You may use the following (unmarked) materials when making this test:
  - Module manual.
  - Slides of the lectures.
  - The book
    David J. Eck. *Introduction to Programming Using Java*. Version 8.1, July 2019.
  - A dictionary of your choice.

  The module manual, the book and the slides can be found at `https://www.utwente.nl/en/telt/learning/intranet/books/`
- 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. You do *not* have to add annotations or comments, unless explicitly asked to do so. Invariants, preconditions and postconditions should be given only when they are explicitly asked.
- No points will be deducted for minor syntax issues such as semicolons, braces and commas in written code, as long as the intended meaning can be made out from your answer.
- This test consists of 6 exercises for which a total of 100 points can be scored. The minimal number of points is zero. Your final grade of this test will be determined by the sum of points obtained for each exercise.

Question 1 (20 points)

In this test, we consider the interfaces, classes and methods necessary to implement an application for managing real state offers.

a. *(5 points)* Define an interface called `Property` with methods to access the following attributes of a *real estate property* (e.g., a house or an apartment): *price*, *address*, *number of rooms*, *living area* and *total area*. Describe minimal reasonable preconditions and postconditions for these methods and explain your choices in the comments.

b. *(5 points)* Suppose an *apartment* is defined as a real estate property where the living area is equal to the total area (i.e., no garden, like a flat apartment). Implement a class called `Apartment` that implements `Property` and define a suitable constructor for this class.

c. *(5 points)* You should know that a **`public`** instance variable can be accessed and modified by the entire program. What is the visibility scope of an instance variable with the **`protected`** modifier, i.e., which objects are allowed to access this variable?

d. *(5 points)* A *studio* is defined as an apartment with a single room. Implement a class called `Studio` that extends `Apartment` and define a constructor for this class.

Question 2 (25 points)

Below you can find a class `Portfolio` that keeps track of real estate offers:

```
public class Portfolio {
    private Set<Property> offers = new HashSet<Property>();
    private int value;

    /** Returns all properties from the offer. */
    public Set<Property> getOffers() {
        return offers;
    }

    /** Returns the value of the portfolio. */
    public int getValue() {
        return value;
    }

    /**
     * Returns a map containing the properties grouped according
     * to the number of rooms.
     */
    public Map<Integer, Set<Property>> groupInRooms() {
        // To be implemented
    }

    /**
     * Returns a sorted list of Properties in this portfolio
     * from low to high price.
     */
    public List<Property> sortByPrice() {
        // To be implemented
    }

    public void addProperty (Property p) {
        offers.add(p);
        value += p.getPrice();
    }
}
```

a. *(10 points)* Implement the method `groupInRooms`. This method returns a `Map` relating the number of rooms to the properties, i.e., given a key `rooms` representing the number of rooms, with `groupInRooms().get(rooms)` all properties with `rooms` rooms can be obtained.

b. *(8 points)* On page 420 of Eck's book (Section 8.5) you can find the `simpleBubbleSort` algorithm to sort an array. Implement the method `sortByPrice`, which returns an `ArrayList` with the contents of the portfolio, but where the portfolio is sorted by price from lowest to highest. You are expected to use methods like `get(int i)` and `set(int i, E element)` of the `List<E>` interface to move elements around according to the algorithm. On page 499 of Eck's book you can find information on these methods.

c. *(7 points)* Write a JUnit test `testSortByPrice` for the method `sortByPrice` that tests whether the result list is sorted and that each offer in the portfolio is also in the sorted result list. Your test case should detect that the following implementation of `sortByPrice` is incorrect:

```
public List<Property> sortByPrice() {
    List<Property> result = new ArrayList<Property>();
    Property oneProperty = getOffers().iterator().next();
    for (int i=0; i < getOffers().size(); i++) {
        result.add(oneProperty);
    }
    return result;
}
```

Question 3 (15 points)

a. *(7 points)* Add a method `removeProperty(Property p)` to the class `Portfolio` of Question 2, which removes a property from the portfolio, throwing an exception if the property is not found. Define a new exception for this purpose. You can use the `contains(Object o)` and `remove(Object o)` methods of the `Collection<E>` interface to implement this method. On page 492 of Eck's book (Section 10.1.4) you can find information on these methods.

b. *(8 points)* Add a method to the `Portfolio` class with the following signature

```
public Set<Property> removeProperties(Set<Property> ps)
```

that tries to remove all properties in `ps` from the portfolio, and returns a set with the properties that were not found in the portfolio and therefore have not been removed. Your method *must* use (i.e., catch) the exception you defined above, otherwise it is be considered incorrect.

Question 4 (15 points)

The following Java interface makes it possible to select properties of a portfolio based on wishes of the customer:

```
public interface CustomerWish {
    /** Returns all properties of Portfolio p that match
     * the wishes of the customer */
    public Set<Property> matches(Portfolio p);
}
```

a. *(5 points)* Implement a class `AreaWish` that implements the `CustomerWish` interface by internally keeping a minimal and a maximal total area (as instance variables) for selecting the properties, so that a property only satisfies the `AreaWish` if its total area is between these bounds.

b. *(5 points)* Implement a method `bestPricedProperties` that, given a `Set` of properties, returns a `Set` of those properties with the lowest price of all the properties in the given `Set`.

c. *(5 points)* Implement a class `CombinedWish` that implements the `CustomerWish` interface by keeping a set of wishes in a `Set<CustomerWish> wishes` field, so that a property only satisfies the `CombinedWish` if it satisfies all the wishes in the `wishes` attribute.
Hint: use the `retainAll` method that is discussed on page 492 of Eck's book (Section 10.1.4).

Question 5 (10 points)

Suppose now that a multithreaded application that uses class `Portfolio` is implemented so that different concurrent threads can make calls to the methods of this class simultaneously. For example, these threads may try to add properties with `addProperty` or remove properties with `removeProperty`.

a. *(5 points)* Explain and give an example of what can go wrong if two threads try to add or remove a property at the same time in this program.

b. *(5 points)* Give a solution to this problem and explain how and why this solution works.

Question 6 (15 points)

Modern houses will have more and more fancy sensors (Internet-of-Things!) to make living more pleasant. Now consider a seller of WiFi-connected digital temperature sensors. The idea is that consumers buy those sensors, connect them to a WiFi network, and go to a web-based online service to view the (graphs of the) temperature measurements from anywhere in the world. Suppose the seller wants to make sure that the messages sent from the sensor to the web server are not tampered with in their travel over the evil internet.

a. *(1 point)* Which of the following methods is most suitable to protect the *integrity* of the messages?

   A. Base64
   B. HMAC
   C. Scrypt
   D. SHA256

b. *(2 points)* Explain your answer to Question 6.a.

c. *(2 points)* Integrity is one of the three often used security properties that, when violated, indicate there is a security incident. What are the other two properties?

Of course the online service for viewing the temperature values has an account system: users can create an account and protect it with a password.

d. *(3 points)* It is common practice to first apply a hash function to a password before storing it. Explain why it is a good a idea for sites to apply a hash function to their users' passwords instead of storing them as-is.

Sometimes users forget their passwords and for this reason this online service has implemented a password-reset functionality. After filling in an e-mail address, users are sent a new password by email. You notice that these reset passwords are 9-11 characters long and have the following pattern:

- First, all passwords start with the text "tt", "bbb", or "gggg",
- after which come 3 characters from the set {"g", "k", "l", "o"} and
- this is followed by four digits.

e. *(2 points)* If an attacker would try to brute-force access to an account with such a reset password, how many attempts would it at most take? Show your calculation.

After a while, the owner of the service is starting to hear rumours that there is an easy way to get access to any account on the system. You are asked to investigate and solve this problem. You look around in the messy code base and find the code shown below. Apparently someone thought it would be a good idea to add some flexibility in the code handling the login process by combining the password with a string that describes which (cryptographic) hash-function should be used. So a password "bike" in combination with the MD5 hash function will be passed along as "MD5:bike".

```
private Map<String, String> passwordDB;

/**
 * Generates the hex-encoded hash of the password using a very flexible
 * scheme. The hash-function to use is embedded in the password: it is
 * separated by a colon. Example passwords: "MD5:abcd" or "SHA1:s3cr3t".
 * @throws NoSuchAlgorithmException
 */
public String getPWHash(String password) throws NoSuchAlgorithmException {
    String[] r = password.split(":");
    String prefix = r[0];
    String realPassword = r[1];
    MessageDigest md = MessageDigest.getInstance(prefix);
    md.update(realPassword.getBytes());
    byte[] digest = md.digest();
    return Hex.encodeHexString(digest);
}

public boolean login(String username, String password) {
    boolean result = true;
    if (passwordDB.containsKey(username)) {
        try {
            String passwordHash = getPWHash(password);
            if (!passwordDB.get(username).equals(passwordHash)) {
                result = false;
            }
        } catch (Exception e) {
            // Whatever, shouldn't happen, right?
        }
    } else {
        result = false;
    }
    return result;
}
```

De Javadoc documentation of the method `String.split` is the following:

`public String[] split(String regex)`

Splits this string around matches of the given regular expression.

This method works as if by invoking the two-argument split method with the given expression and a limit argument of zero. Trailing empty strings are therefore not included in the resulting array.

The string "boo:and:foo", for example, yields the following results with these expressions:

Regex	**Result**
`:`	`{ "boo", "and", "foo" }`
`o`	`{ "b", "", ":and:f" }`

Parameters

regex - the delimiting regular expression

Returns

the array of strings computed by splitting this string around matches of the given regular expression

Suppose an attacker (somehow) has full control over the *content* of the variables `username` and `password` that are passed along to the `login` method. There are (at least) two ways for the attacker to get the `login` method to return true (and thus gaining access) without actually knowing a password.

f. *(5 points)* Describe at least one way an attacker can get access to any account. Also show how one can (easily) solve this issue in the code.