nexthink

Nexthink Java Quiz

Thank you very much for filling out this quiz to the best of your competences. The purpose of this quiz is to evaluate your level of expertise in technologies and techniques that we use daily @Nexthink.

Good luck and thank you for your time!

All the source code for this challenge can be found under the following link:

https://github.com/sparlampe/nexthink-coding-challenge

Question 1:

A producer thread periodically produces elements and puts them in a queue, and a consumer thread takes the elements from the queue and does some processing with them:

```
private final List<Integer> queue = new LinkedList<>();
```

Consumer

```
synchronized (_____) {
  while (_____) {
    _____
  }
  Integer num = queue.remove(0);
  _____
  processElement(num);
}
```

Producer

```
synchronized (_____) {
  int maxSize = 1000L;
  while (_____) {
    _____
  }
  queue.add(new Random().nextInt());
  _____
}
```

Please write the missing instructions for the producer and consumer threads. Feel free to modify the given code structure, if needed.

Please see the implementation under the following link.

Question 2:

Please complete the “sum” method in the following code (in a recursive way):

```
public class Payment {
   private final int amount;

   public Payment(int amount) {
      this.amount = amount;
   }

   public int getAmount() {
      return amount;
   }
}

public class Question3 {

   public static void main(String[] args) {
      List<Payment> payments = new ArrayList<>();
      payments.add(new Payment(100));
      payments.add(new Payment(50));

      System.out.println(sum(…));
   }

   private static int sum(…) {…}
}
```

Recursive sum:

Please see the implementation under the following link.

What can go wrong with this recursive function if the number of payments becomes too big?

Each recursive call adds information on the stack. A very deep recursion can overflow the stack.

Question 3:

Given the following code:

```
public class SafeLock {
   private Lock lock = new ReentrantLock();

   public void lock() throws InterruptedException {
      LockingManager.registerLock(this);
      long timeout = 1000;
      while (!lock.tryLock(timeout, TimeUnit.MILLISECONDS)) {
         if (LockingManager.checkForCircularDependency()) {
            throw new ThreadLockException();
         }
      }
   }

   public void unlock() {
      LockingManager.unregisterLock(this);
      lock.unlock();
   }
}

public class ThreadLockException extends RuntimeException   {
}
```

This class is designed to cope with a problematic situation when dealing with locks, what is it?

The code tries to deal with thread deadlocks. For example, thread 1 locked resource 1 and waits for a opportunity to lock resource 2. Meanwhile thread 2 already locked resource 2 and waits for an opportunity to lock resouce1. In this situation both treads a locked into waiting.

What techniques can be used to avoid running into such a situation?

1. Using some other concurrency model, for example actors.
2. Requesting locks in the same order from any thread.
3. Timing out after waiting for a lock for threshold time.

What does checkForCircularDependency() possibly do?

The function will try to figure out if the thread did not get the lock because of circular locking attempts like in a thread deadlock situation or for a transient reason. In case of a transient lock rejection, the thread will retry. See my attempt at implementing the missing pieces under the following link.

What particular characteristic do the static methods in LockingManager require to work properly?

The methods should be mutually exclusively executable by threads, synchronized.

Question 4:

When running the following program:

```
class Animal {
   private final int id;
   private final String name;

   public Animal(int id, String name) {
      this.id = id;
      this.name = name;
   }

   @Override
   public boolean equals(Object other) {
      if (this == other) return true;
      if (!(other instanceof Animal)) return false;
      Animal otherAnimal = (Animal) other;
      return id == otherAnimal.id;
   }
}

public class Question5 {
   public static void main(String[] args) {
      Animal lion1 = new Animal(1, "lion");
      Animal lion2 = new Animal(1, "lion");
      Set<Animal> animals = new HashSet<>();
      animals.add(lion1);
      animals.add(lion2);
      System.out.println("Number of animals: " + animals.size());
   }
}
```

The result is:

```
Number of animals: 2
```

The result should instead be `Number of animals: 1`. What is missing in the Animal class? Please also provide an implementation of the missing part.

HashCode is not properly implemented. See the implementation under the following link.

Question 5:

Consider the code presented in folder Question5 of the attached Java resources.

The program is a simple expression evaluator able to compute integer expressions bases on four operators (add, subtract, multiply and divide). The logic is implemented in class *Expression*, which offers the following features:

- Evaluating the expression to retrieve its Integer value;
- Pretty printing the expression;
- Exporting the expression to XML.

The goal of this exercise is to refactor the *Expression* class (and optionally the main program) to improve the code. You can change the code in any way you want, keeping in mind that:

- In the main program, we still want to create expression programmatically (no need to implement a parser);
- The 3 features described above must behave like in the original code, specifically:
    - No matter how the API for constructing the expression is implemented, it must still take care of invalid input.
    - The output of pretty-printing the expression must be the same as now.
    - The output of XML serialization must be the same as now.
- You can add classes and change the API as you wish, as long as the previous points are not violated.

What we're looking for is clean, solid, easy to understand, and correct code. Also think about which design pattern can help you make the code more readable and avoid code duplication.

See the implementation under the following link.

Nexthink React Quiz

Question 6:

Given the following code:

```
for (var i = 0; i < 5; i++) {
    var btn = document.createElement('button');
    btn.appendChild(document.createTextNode('Button ' + i));
    btn.addEventListener('click', function(){ console.log(i); });
    document.body.appendChild(btn);
}
```

What is the result when you click on Button 3?

The value is 5.

Explain why and mechanisms that are behind?

var declares i the parent function scope of the for-loop. The event listener function chains its scope to the function scope enclosing the for-loop, where i is declared. When the event listener executes, the scope lookup for i end up at the variable in the upper function scope that by the end of the for-loop execution has value 5.

Please see possible solutions by either creating an additional function scope (works for old ECMAScript) or by using newer **let** declaration that induces creation of scope for each loop iteration.

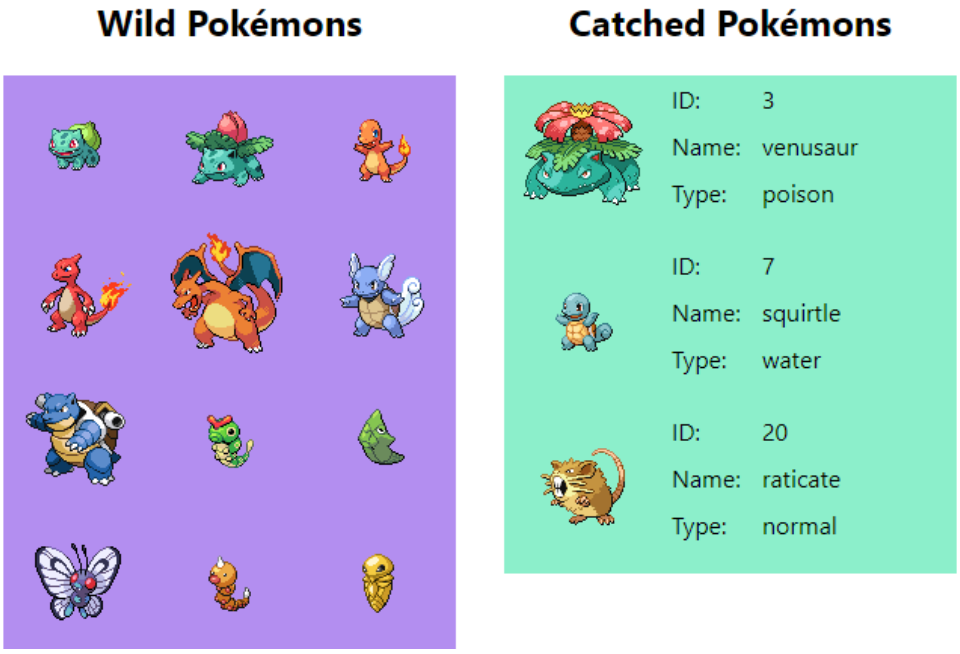
Source code under the following link.

Live demo under the following link.

Question 7:

Create a React project which allows the user to catch and release Pokémons.

You will have two lists, **wild Pokémons** and **catched Pokémons.**



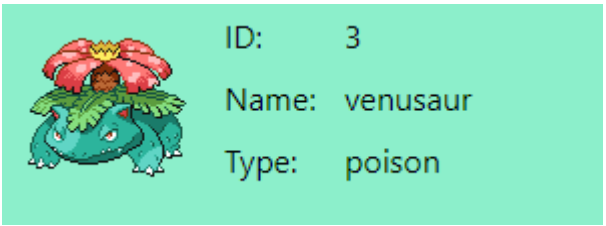
Wild Pokémons

- Display a list of 0 to 151 Pokémons (ID from 1 to 151).
- A new Pokémon must appear in the list every half second until all 151 Pokémons are loaded.
- The items should be ordered by Pokémon Id.
- Pokémons will be displayed as follow (Here you can see them being loaded one by one).

- The list will display at maximum 3 Pokémons horizontally.
- The list can expand vertically as much as needed to fit all the Pokémons.
- Clicking on a wild Pokémon will remove it from the **wild Pokémons** list and put it in the **catched Pokémons** list.
- There can be at maximum 6 **catched Pokémons**, when the limit is reached, clicking on a wild Pokémon has no action.

Catched Pokémons

- Display a list of 0 to 6 Pokémons. One item horizontally, displayed as followed.



- Clicking on a catched Pokémon will release it into the wild by adding it back to the wild list and removing it from the catched list.

API

To get the Pokémons you will have to query an API. https://pokeapi.co/docs/v2.html#pokemon

You will need to use a GET query on the following.

```
https://pokeapi.co/api/v2/pokemon/id  //  id  of  pokemon  is
integer
```

With id being the id of the Pokémon. You'll get a JSON response which you can parse as following:

```
id = data.id;
name = data.name;
sprite = data.sprites.front_default; // Url of image
type = data.types[0].type.name;
```

Guidelines

This project should be a static web page and could be executed from files, no need of persistence or any server.

- Use the library you prefer and explain your choices (what it brings to you, advantages or inconvenient)
- Explain how you can test this project
- Language can be Javascript or Typescript
- Gotta Catch 'Em All!

Please the source code under the following link.

Try out the live version under this link.

I have made the following choices:

- **pokeapi-typescript** for typing of the pokeipi
- **react** as virtual dom library to avoid working with dom directly, but rather to work with a higher level abstraction.
- **rxjs** for state management in functional reactive programming style. In my view the style allows creating highly interactive UIs with much less overhead than other popular alternatives.
- **rxjs-hooks** for interfacing react hooks of the components with observables.