2690. Infinite Method Object

Problem Description

Easy

The problem requires us to create a function that constructs and returns an infinite-method object. An infinite-method object is a special type of object that can handle calls to any method, even if that method was not explicitly defined on the object. Whenever any method is called on this object, it should return the name of that method as a string.

For instance, if you have an object named obj and you call obj.someMethodName(), it should return the string "someMethodName". This behavior should apply universally to any method name provided when calling the object.

This problem tests one's understanding of dynamic properties and methods in object-oriented programming, as well as the

implementation of such behavior in the specific programming language required by the problem, which in this case is TypeScript. Intuition

To create this infinite-method object, we make use of the Proxy object which is a feature in JavaScript and TypeScript that allows us to define custom behavior for fundamental operations (e.g., property lookup, assignment, enumeration, function invocation,

etc.). By creating a Proxy, we can control the behavior of the object when a method is called that does not exist on the object. The intuition behind the solution is to capture all method calls on the fly and return a function that, when invoked, returns the name of the non-existent method. The get trap in the Proxy is useful for this purpose. This trap will fire every time a property on

the target object is accessed. Here's how we can break down the key points of the solution:

1. We create and return a Proxy object. 2. The target object of the Proxy is an empty object {}.

- 3. Whenever any property (in our use case, a function name) is accessed on the Proxy object, the get trap is triggered. 4. The get trap is a function that takes the target object and the property being accessed (in our case, the method name).
- 5. The get trap returns a new arrow function.
- 6. This arrow function, when called, returns the name of the property accessed, which corresponds to the method name.
- **Solution Approach**
- The implementation of the solution follows a distinct approach using a design pattern called Proxy, which is inherently provided in
- JavaScript and TypeScript languages. The Proxy pattern creates an object that wraps around another object and intercepts its fundamental operations, such as property access or function invocation. Here's how the solution is achieved step by step:

Create a Proxy: The solution defines a function createInfiniteObject() that returns a new Proxy object.

accessed.

that property as a string.

Define the target object: The Proxy is initialized with an empty object {} as its target. This object would normally not have any properties or methods. **Define the Proxy handlers:** A handler object is provided as the second argument to the Proxy constructor. This handler object

- contains the get trap, a function that will be invoked every time a property on the Proxy (which includes methods) is
- the name of the property that is being accessed. Since we want to capture function calls, prop will be the name of the method being invoked. **Return a function dynamically**: Inside the get trap, we return an arrow function () => prop.toString(). This ensures that no

matter what property name (or method name) is accessed, a function is returned that, when called, will return the name of

Implement the get trap: The get trap is a function that accepts two parameters: target, which is the target object, and prop,

- Leverage anonymous functions: Every time a method on the Proxy is called, like obj.someMethod(), the get trap executes and returns a new anonymous function that contains the code to simply return the prop as a string.
- expected name of the method without explicitly defining those methods. It's a powerful way to create objects with behavior that is determined at runtime rather than compile-time. This opens up possibilities for dynamic and flexible code structures. Here is the code of the function for reference:

By following these steps, we harness the power of the Proxy pattern to dynamically intercept method calls and return the

},

When this function is used to create an object, it will inherently have the described infinite-method behavior, as desired.

To better understand the solution approach described above, let's walk through a small example using the

Example Walkthrough

console.log(obj.helloWorld());

property access.

Following the steps of the solution approach:

return new Proxy(

{},

```
createInfiniteObject() function.
  First, we call createInfiniteObject() to create a new object:
let obj = createInfiniteObject();
  At this point, obj is an instance of a Proxy object that is set to respond to any method invocation using the rules defined in the
```

get trap. Let's see what happens when we try to call a method that hasn't been defined:

function createInfiniteObject(): Record<string, () => string> {

get: (_, prop) => () => prop.toString(),

The get trap function is called with the target (which is an empty object) and the prop (which is the string "helloworld"). Inside the get trap, it returns an anonymous arrow function: () => prop.toString(). 3.

This behavior illustrates that any method name we try to call on obj will result in the name of the method being returned as a

In each case, the Proxy intercepts the method call, the get trap provides an anonymous function, the function is called, and the

method name is returned as a string. Through this example, we've seen first-hand how the Proxy pattern allowed us to handle

This arrow function is immediately invoked (because we have used parentheses () after obj.helloworld).

We attempt to access the property helloworld on obj. Since obj is a Proxy, the get trap is triggered instead of a typical

console.log(obj.yetAnotherOne()); // Outputs: "yetAnotherOne" console.log(obj['any.method']); // Outputs: "any.method"

The console. log statement outputs the string "helloworld" to the console.

The invoked arrow function returns the string "helloworld".

string. It works exactly the same for any other method name:

console.log(obj.anotherMethod()); // Outputs: "anotherMethod"

Solution Implementation

dynamic method calls in a generic way without ever defining any actual methods on the object.

When this function is called, it needs to return the name of the property.

This behavior will be defined in the InfiniteObject to ensure it returns

This method is called when an attribute is accessed that doesn't exist

in the object's dictionary. It returns a callable that returns the attribute's name.

:return: A callable object that returns the name of the property when called.

This function will return the name of the property when it is called.

Define a function that will act as the StringReturningFunction callable.

def __getattr__(self, name: str) -> StringReturningFunction:

:param name: The name of the attribute being accessed.

def string_returning_function() -> str:

return string_returning_function

print(infinite_object.abc123()) # Outputs: "abc123"

Usage example of InfiniteObject class

import java.lang.reflect.InvocationHandler;

infinite_object = InfiniteObject()

import java.lang.reflect.Method;

import java.util.function.Supplier;

import java.lang.reflect.Proxy;

import java.util.HashMap;

@Override

#include <unordered_map>

class InfiniteObject {

};

int main() {

return 0;

return new Proxy(

the correct property name.

return name

Usage example of InfiniteObject class

infinite_object = InfiniteObject()

return string_returning_function

pass

public:

};

/**

// Define the InfiniteObject class

return null;

import java.util.Map;

the correct property name. # The actual implementation will be provided in the InfiniteObject.__getattr__ method. pass

```
:return: The name of the property as a string.
    return name
# Return the above-defined function as a StringReturningFunction.
```

Java

/**

*/

/**

Python

class StringReturningFunction:

class InfiniteObject:

def __call__(self) -> str:

```
interface InfiniteObject {
   StringReturningFunction get(String property);
/**
* Creates a proxy object that dynamically generates methods such that any method call will return
* a StringReturningFunction that, when called, returns the name of the method as a String.
* @return An InfiniteObject with dynamically generated methods.
public static InfiniteObject createInfiniteObject() {
   // Create an InvocationHandler that defines the behavior of the proxy instance.
```

InvocationHandler handler = new InvocationHandler() {

// and takes one argument of type String.

String propertyName = (String) args[0];

interface StringReturningFunction extends Supplier<String> {

* The InfiniteObject interface acts similarly to a map,

* A functional interface that represents a function that returns a String.

* where the methods (properties in JavaScript) return StringReturningFunction instances.

// The invoke method is called whenever a method on the proxy instance is invoked.

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {

if ("get".equals(method.getName()) && args.length == 1 && args[0] instanceof String) {

// The lambda function captures the property name and returns it when called.

// We return a lambda function if the invoked method is 'get'

return (StringReturningFunction) () -> propertyName;

```
};
    // Create and return the proxy instance that implements the InfiniteObject interface.
    return (InfiniteObject) Proxy.newProxyInstance(
       InfiniteObject.class.getClassLoader(),
       new Class<?>[]{InfiniteObject.class},
       handler
/**
* Usage example of the createInfiniteObject method.
public static void main(String[] args) {
    // Create an infinite object instance.
    InfiniteObject infiniteObject = createInfiniteObject();
    // Access the properties using the get method and execute the returned function.
    System.out.println(infiniteObject.get("abc123").get()); // Outputs: "abc123"
C++
#include <iostream>
#include <string>
#include <functional>
```

// This operator overloads the '[]' operator to provide the ability to access string properties

std::string result = infiniteObject[propertyKey](); // This calls the lambda function and outputs: "abc123"

// Return a lambda that captures the property key and returns it when called

```
TypeScript
// Define a type that represents a function that returns a string
type StringReturningFunction = () => string;
// Define the InfiniteObject type as a Record where the keys are strings
```

// and the values are functions that return strings.

* returns the name of the property as a string.

function createInfiniteObject(): InfiniteObject {

type InfiniteObject = Record<string, StringReturningFunction>;

* @returns {InfiniteObject} An object with infinite properties using a Proxy.

// property access, assignment, enumeration, function invocation, etc.

{}, // The target object, which is an empty object in this case.

// Define a function pointer that returns a std::string

return [propertyKey]() -> std::string {

return propertyKey;

// Usage example of InfiniteObject class

std::string propertyKey = "abc123";

std::cout << result << std::endl;</pre>

InfiniteObject infiniteObject;

using StringReturningFunction = std::function<std::string()>;

StringReturningFunction operator[](const std::string& propertyKey) {

// When the function is called, it returns the property key

// The overloaded '[]' operator returns a lambda function that returns the key

```
// The handler object, containing traps for various operations.
              // The 'get' trap is used to intercept property access.
              get: (_, propertyKey) => {
                  // Return a function that, when called, returns the property key as a string.
                  // The '_' is a commonly used convention to indicate that the parameter is not used.
                  return () => propertyKey.toString();
    Usage example of createInfiniteObject function (uncomment to execute)
  // const infiniteObject = createInfiniteObject();
  // console.log(infiniteObject['abc123']()); // Outputs: "abc123"
class StringReturningFunction:
   def __call__(self) -> str:
```

When this function is called, it needs to return the name of the property.

This behavior will be defined in the InfiniteObject to ensure it returns

* Creates an object that returns functions from property access, where the function, when called,

// Return a Proxy object. Proxies enable creating objects with custom behavior for

```
class InfiniteObject:
   def __getattr__(self, name: str) -> StringReturningFunction:
       This method is called when an attribute is accessed that doesn't exist
        in the object's dictionary. It returns a callable that returns the attribute's name.
        :param name: The name of the attribute being accessed.
        :return: A callable object that returns the name of the property when called.
       # Define a function that will act as the StringReturningFunction callable.
       def string_returning_function() -> str:
            This function will return the name of the property when it is called.
```

The actual implementation will be provided in the InfiniteObject.__getattr__ method.

print(infinite_object.abc123()) # Outputs: "abc123" Time and Space Complexity

Return the above-defined function as a StringReturningFunction.

:return: The name of the property as a string.

The time complexity of the createInfiniteObject function call itself is O(1) because it returns a proxy object without performing any computation. When any property on the resulting proxy object is accessed and the function returned is called, it performs this action in constant time, resulting in a time complexity of O(1) per property access.