Yandex

Universal Memoization Decorator

Victor Komarov
Software Engineer

 victor-k@yandex-team.ru

 +7 929 591-09-69

Memoization

Python

```
def DoHeavyStuff(x, y, z):
    ...
    return result
```

Python

```
@memoize
def DoHeavyStuff(x, y, z):
    ...
    return result
```

C++

```
int DoHeavyStuff(int x, const std::string& y, double z) {
    ...
    return result;
}
```

C++

```
int DoHeavyStuff(int x, const std::string& y, double z) {
    ...
    return result;
}

using F = Memoize<DoHeavyStuff, HashMap>;
auto val = F(x, y, z);
```

How?

Cache container

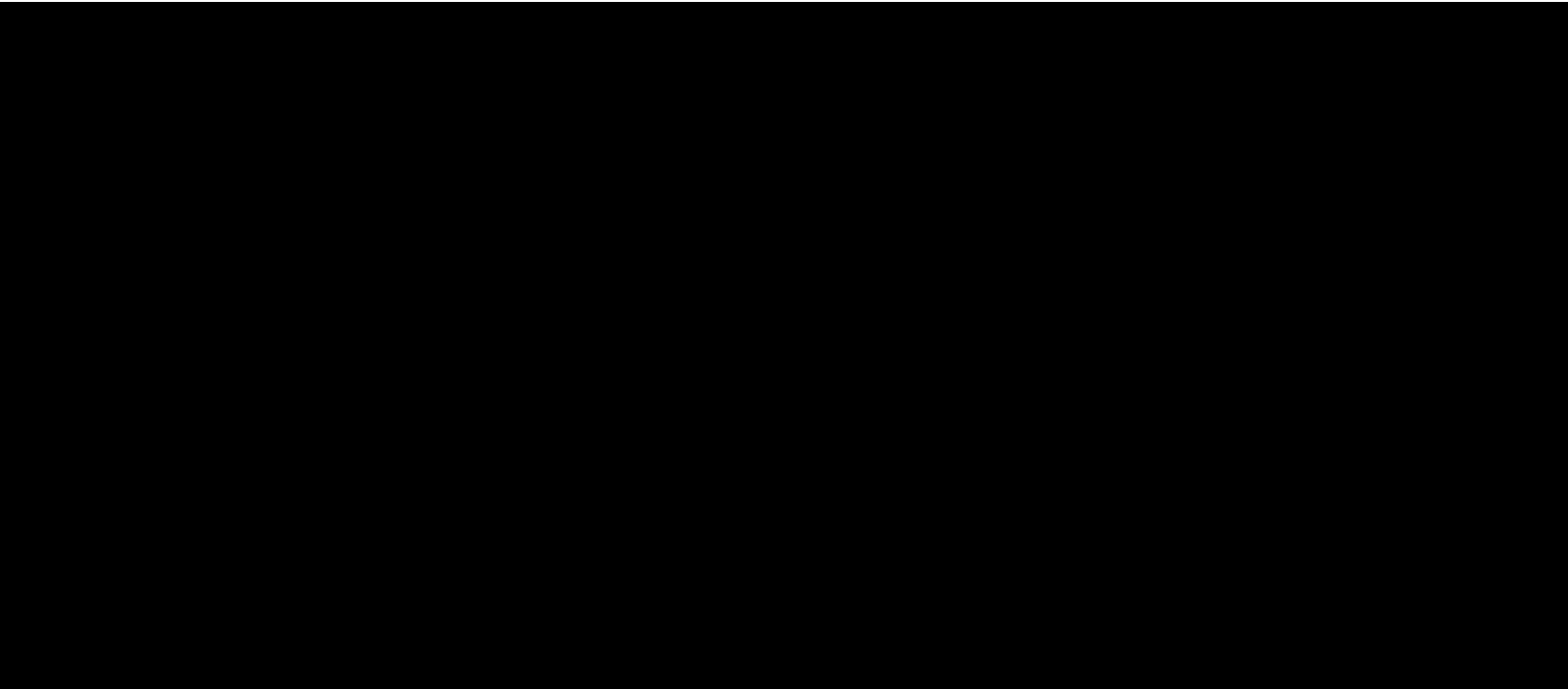
```
TValue f(TArgs...);

using TKey = std::tuple<typename std::decay<TArgs>::type...>;

template<typename TKey, typename TValue>
class HashMap {
public:
    void set(const TKey& key, const TValue& value);
    const TValue* get(const TKey& key) const;
    ...
}
```

Each instantiation has its own static variables

What about collisions?



What about collisions?

```
template<typename T, T t>
```

```cpp
template<typename T, T t>
class A {
public:
    static void f() {
        static int val = 0;
        std::cout << size_t(&val) << std::endl;
    }
};

int main() {
    A<int, 0>::f();
    A<int, 1>::f();
    A<int, 2>::f();
    return 0;
}
```

4457812440

4457812444

4457812448

Building the decorator

```
template<typename T, T t,
         template<typename, typename> class TCacheContainer>
class Memoize;
```

Building the decorator

```
template<typename T, T t,
         template<typename, typename> class TCacheContainer>
class Memoize;

template<typename TValue, typename... TArgs, TValue (*f)(TArgs...),
         template<typename, typename> class TCacheContainer>
class Memoize<TValue (*)(TArgs...), f, TCacheContainer> {
  ...
```

```
static TValue call(TArgs... args) {
    static TCacheContainer<TKey, TValue> cache;
    auto key = TKey(args...);
    auto result = cache.get(key);
    if (result != nullptr) {
        return *result;
    } else {
        auto value = f(args...);
        cache.set(key, value);
        return value;
    }
}
```

Usage

Example

```
int DoHeavyStuff(int x, const std::string& y, double z) {
    ...
    return result;
}

using F = Memoize<decltype(&DoHeavyStuff), &DoHeavyStuff, HashMap>;
auto val = F::call(x, y, z);
```

Fibonacci Example

```
int Fibonacci(int n);

int FibonacciImpl(int n) {
    if (n <= 1) return 1;
    return Fibonacci(n - 1) + Fibonacci(n - 2);
}

int Fibonacci(int n) {
    using F = Memoize<decltype(&FibonacciImpl), &FibonacciImpl, HashMap>;
    return F::call(n);
}
```

What’s left behind

1. Interaction with lambdas, std::function, std::bind etc.
2. Other containers: hash maps, fixed-sized maps, LRU
3. Concurrency

Thank you!