

Dynamic Proxies in Java

Dynamic Proxies in Java

Dr Heinz M. Kabutz

Last updated 2020-07-15

© 2020 Heinz Kabutz – All Rights Reserved



Javaspecialists.eu
java training

whois Heinz?

tinyurl.com/nairobi-jug

- **Not ketchup**
- **10+ years teaching remotely from Crete**
 - learning.javaspecialists.eu
 - Threading and concurrency, patterns, advanced topics, etc.
 - Contact: heinz@jaspecialists.eu
- **Java Champion, Speaker, bla bla**



Dynamic Proxies in Java

[tinyurl.com/
nairobi-jug](http://tinyurl.com/nairobi-jug)



Javaspecialists.eu
java training

History of Dynamic Proxies

- **RMI used to need a separate compile step**
 - Tool "rmic" still found in JDK/bin directory
 - Creates *stubs* and *skeletons* to manage remote method invocations
- **Java 1.3 released in May 2000**
 - First version with dynamic proxies
 - Functional interface of `InvocationHandler` to service all methods on proxy
 - Not necessary to use "rmic" or similar tools for deployment
 - Made it possible to build flexible, dynamic systems

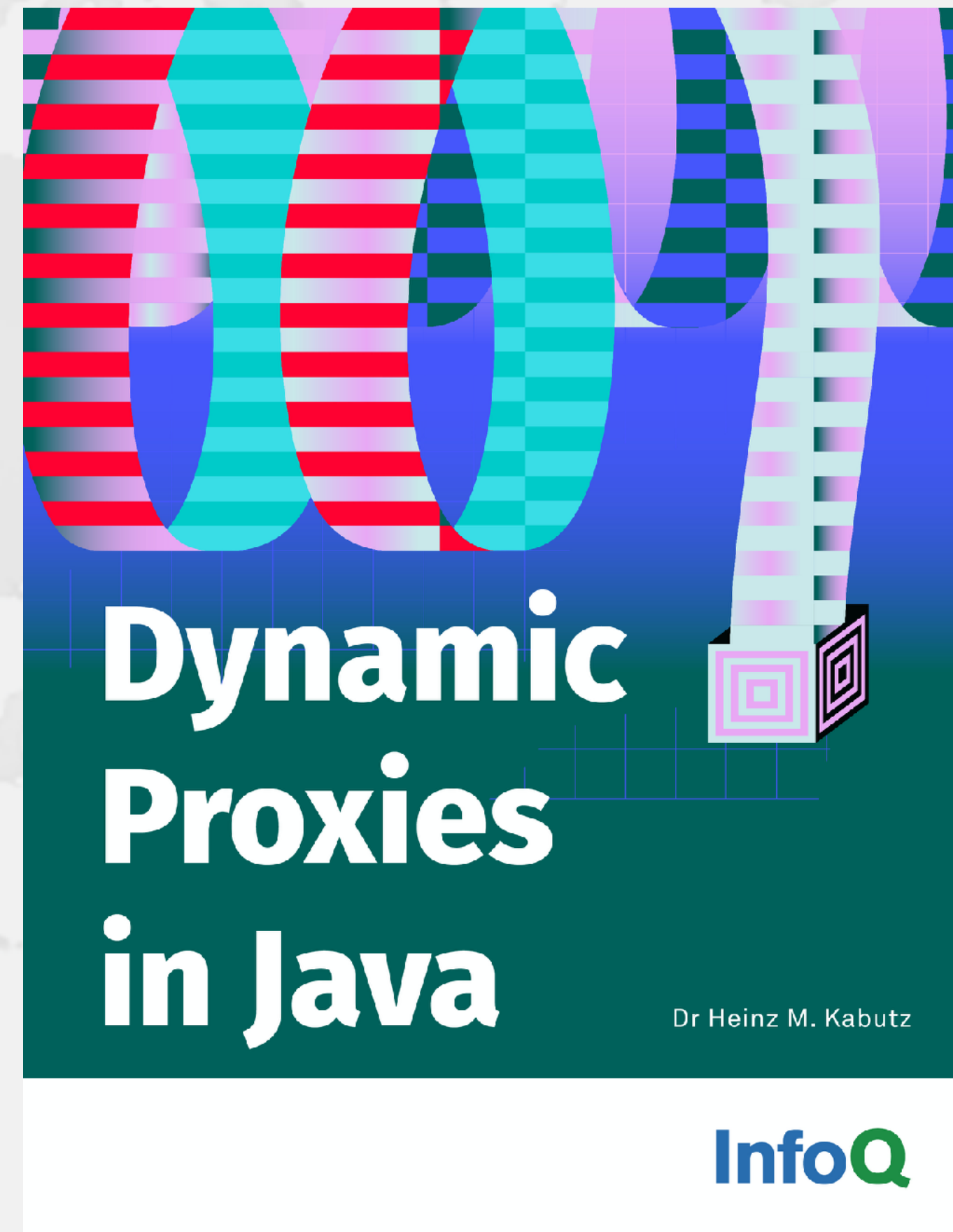
Big Win

- **Don't Repeat Yourself (DRY) at its best**
 - Write a single InvocationHandler implementation
 - Reuse for hundreds of classes
- **We once replaced 600,000 code statements with 1 dynamic proxy**
 - Code had been generated, but was maintained by hand
 - Dynamic proxy easier to maintain
 - Less code

Infrastructure Code

- **Dynamic proxies in tools and frameworks**
 - **Spring**
 - **Annotations**
 - **Dependency injection**
 - **Hibernate**
 - **Gradle**

Dynamic Proxies in Java



- **Free download from**
 - www.infoq.com/minibooks/java-dynamic-proxies

1: Handcrafted Proxies



Handcrafted Proxies

- **Before learning how to avoid duplicate code, we will copy & paste**
 - **And then in next section will use *dynamic proxies* instead**

Dynamic Proxies in Java

Virtual Proxy



Javaspecialists.eu
java training

Virtual Proxy

- **Delays expensive object creation**
 - placeholder object creates costly object on demand

CustomMap Interface

- **Reduced version of the Map interface**

```
public interface CustomMap<K, V> {  
    int size();  
    V get(Object key);  
    V put(K key, V value);  
    V remove(Object key);  
    void clear();  
    void forEach(BiConsumer<? super K, ? super V> action);  
}
```

CustomHashMap Implementation

- **Delegates methods to a java.util.HashMap**
 - Repetitive and error prone

```
public class CustomHashMap<K, V> implements CustomMap<K, V> {  
    private final Map<K, V> map = new HashMap<>();  
    { System.out.println("CustomHashMap constructed"); }  
    public int size() { return map.size(); }  
    public V get(Object key) { return map.get(key); }  
    public V put(K key, V value) { return map.put(key, value); }  
    public V remove(Object key) { return map.remove(key); }  
    public void clear() { map.clear(); }  
    public void forEach(BiConsumer<? super K, ? super V> action) {  
        map.forEach(action);  
    }  
    public String toString() { return map.toString(); }  
}
```

VirtualCustomMap Virtual Proxy

- **Has a reference to a Supplier for CustomMap**
 - Is created in the `getRealMap()` method

```
public class VirtualCustomMap<K, V>
    implements CustomMap<K, V> {
    private final Supplier<CustomMap<K, V>> mapSupplier;
    private CustomMap<K, V> realMap;
    public VirtualCustomMap(
        Supplier<CustomMap<K, V>> mapSupplier) {
        this.mapSupplier = mapSupplier;
    }
    private CustomMap<K, V> getRealMap() {
        // not thread-safe
        if (realMap == null) realMap = mapSupplier.get();
        return realMap;
    }
}
```

VirtualCustomMap Methods

```
public int size() {
    return getRealMap().size();
}
public V get(Object key) {
    return getRealMap().get(key);
}
public V put(K key, V value) {
    return getRealMap().put(key, value);
}
public V remove(Object key) {
    return getRealMap().remove(key);
}
public void clear() { getRealMap().clear(); }
public void forEach(
    BiConsumer<? super K, ? super V> action) {
    getRealMap().forEach(action);
}
}
```

Using VirtualCustomMap

- **CustomHashMap made when method called**
 - **Does not matter which method we call first**

```
CustomMap<String, Integer> map =  
    new VirtualCustomMap<>(CustomHashMap::new);  
System.out.println("Virtual Map created");  
map.put("one", 1);  
map.put("life", 42);  
System.out.println("get(\"life\") = " + map.get("life"));  
System.out.println("size() = " + map.size());  
System.out.println("clearing map");  
map.clear();  
System.out.println("size() = " + map.size());
```

```
Virtual Map created  
CustomHashMap constructed  
get("life") = 42  
size() = 2  
clearing map  
size() = 0
```


2: Dynamic Proxy



2: Dynamic Proxy

- **Avoid copy and paste programming**
 - A bug needs to be fixed everywhere
- **Better is static or dynamic code generation**

Proxy.newProxyInstance()



Proxy.newInstance()

- **Takes three parameters**
 - **ClassLoader** where the new proxy class is loaded
 - **Class<?>[]** an array containing all interfaces our proxy object must implement
 - **InvocationHandler** a handler that's called when any proxy method is invoked

InvocationHandler

- **Invoked when *any* method is called on proxy**

```
public interface InvocationHandler {  
    Object invoke(Object proxy, Method method,  
                Object[] args) throws Throwable;  
}
```

- Object proxy **is the instance of the dynamic proxy class that is calling invoke()**
- Method method **is a java.lang.reflect.Method**
 - **Either interface method or equals(), hashCode(), or toString()**
- Object[] args **is an array of parameters passed into the method**
 - **This is null when method has no parameters**

Dynamic Proxies in Java

[tinyurl.com/
nairobi-jug](http://tinyurl.com/nairobi-jug)



Javaspecialists.eu
java training

Dynamic Proxies in Java

LoggingInvocation Handler



Javaspecialists.eu
java training

LoggingInvocationHandler

- **We will log all method calls**
 - Optionally measuring how long they take
- **The constructor parameters are**
 - Logger log a `java.util.Logger` to log to
 - Object obj **the object that we want to delegate the calls to**
 - **Must implement the same interfaces as the proxy**

```
public final class LoggingInvocationHandler implements InvocationHandler {  
    private final Logger log;  
    private final Object obj;  
    public LoggingInvocationHandler(Logger log, Object obj) {  
        this.log = log;  
        this.obj = obj;  
    }  
}
```


invoke() Method for Logging

```
public Object invoke(
    Object proxy, Method method, Object[] args)
    throws Throwable {
    log.info(() -> "Entering " + toString(method, args));
    // optimization - nanoTime() is expensive native call
    final boolean logFine = log.isLoggable(Level.FINE);
    long start = logFine ? System.nanoTime() : 0;
    try {
        return method.invoke(obj, args);
    } finally {
        long nanos = logFine ? System.nanoTime() - start:0;
        log.info(() -> "Exiting " + toString(method, args));
        if (logFine) log.fine(() -> "Time " + nanos + "ns");
    }
}
```

toString() Prints Methods with Args

```
private String toString(Method method,
                        Object[] args) {
    return String.format("%s.%s(%s)",
        method.getDeclaringClass().getCanonicalName(),
        method.getName(),
        args == null ? "" :
            Stream.of(args).map(String::valueOf)
                .collect(Collectors.joining(", ")));
}
```

Demo of LoggingInvocationHandler

```
@SuppressWarnings("unchecked")
var map = (Map<String, Integer>)
    Proxy.newProxyInstance(Map.class.getClassLoader(),
        new Class<?>[] {Map.class},
        new LoggingInvocationHandler(
            Logger.getGlobal(), new ConcurrentHashMap<>()));
map.put("one", 1);
map.put("two", 2);
System.out.println(map);
map.clear();
```

```
Jan 24, 2020 7:32:20 AM
eu.javaspecialists.books.dynamicproxies.ch03.logging.LoggingInvocationHandler invoke
INFO: Entering java.util.Map.put(one, 1)
Jan 24, 2020 7:32:21 AM
eu.javaspecialists.books.dynamicproxies.ch03.logging.LoggingInvocationHandler invoke
INFO: Exiting java.util.Map.put(one, 1)
Jan 24, 2020 7:32:21 AM
eu.javaspecialists.books.dynamicproxies.ch03.logging.LoggingInvocationHandler invoke
FINE: Time 61622ns
```

Dynamic Proxies in Java

Dissecting a Dynamic Proxy



Javaspecialists.eu
java training

Dissecting a Dynamic Proxy

- **We will start with a simple interface**

```
public interface ISODateParser {  
    LocalDate parse(String date) throws ParseException;  
}
```

Dynamic Proxy Class Name

- **Dynamic proxy with empty InvocationHandler**

```
System.out.println(  
    Proxy.newProxyInstance(  
        ISODateParser.class.getClassLoader(),  
        new Class<?>[] {ISODateParser.class},  
        (proxy, method, arguments) -> null  
    ).getClass()  
);
```

```
class com.sun.proxy.$Proxy0
```

Decompiling \$Proxy0

- **We can dump generated proxy classes**
 - Java 9+:
 - Djdk.proxy.ProxyGenerator.saveGeneratedFiles=true
 - Earlier versions:
 - Dsun.misc.ProxyGenerator.saveGeneratedFiles=true
- **And then decompile with a tool like CFR**
 - <https://www.benf.org/other/cfr>

Dynamic Proxies in Java

```
public final class $Proxy0 extends Proxy
    implements ISODateParser {
    private static Method m0;
    private static Method m1;
    private static Method m2;
    private static Method m3;
    static {
        try {
            m0 = Class.forName("java.lang.Object").getMethod("hashCode");
            m1 = Class.forName("java.lang.Object").getMethod("equals",
                Class.forName("java.lang.Object"));
            m2 = Class.forName("java.lang.Object").getMethod("toString");
            m3 = Class.forName("eu.javaspecialists.books.dynamicproxies."
                + "ch03.ISODateParser").getMethod("parse",
                Class.forName("java.lang.String"));
        } catch (NoSuchMethodException e) {
            throw new NoSuchMethodError(e.getMessage());
        } catch (ClassNotFoundException e) {
            throw new NoClassDefFoundError(e.getMessage());
        }
    }
}
```


Dynamic Proxies in Java

```
public $Proxy0(InvocationHandler h) {super(h);}

public final int hashCode() {
    try {
        return (Integer) h.invoke(this, m0,
            (Object[]) null);
    } catch (RuntimeException | Error e) {
        throw e;
    } catch (Throwable e) {
        throw new UndeclaredThrowableException(e);
    }
}

public final boolean equals(Object o) {
    try {
        return (Boolean) h.invoke(this, m1, new Object[] {o});
    } catch (RuntimeException | Error e) {
        throw e;
    } catch (Throwable e) {
        throw new UndeclaredThrowableException(e);
    }
}
```

Dynamic Proxies in Java

```
public final String toString() {  
    try {  
        return (String) h.invoke(this, m2,  
            (Object[]) null);  
    } catch (RuntimeException | Error e) {  
        throw e;  
    } catch (Throwable e) {  
        throw new UndeclaredThrowableException(e);  
    }  
}
```

```
public final LocalDate parse(String s) throws ParseException {  
    try {  
        return (LocalDate) h.invoke(this, m3, new Object[] {s});  
    } catch (RuntimeException | ParseException | Error e) {  
        throw e;  
    } catch (Throwable e) {  
        throw new UndeclaredThrowableException(e);  
    }  
}
```

Dynamic Proxies in Java

Virtual Dynamic Proxy



Javaspecialists.eu
java training

Virtual Dynamic Proxy

- **InvocationHandler for virtual proxies**

```
public final class VirtualProxyHandler<S>
    implements InvocationHandler, Serializable {
    private final Supplier<? extends S> supplier;
    private S subject;
    public VirtualProxyHandler(Supplier<? extends S> supplier) {
        this.supplier = supplier;
    }
    private S getSubject() {
        if (subject == null) subject = supplier.get();
        return subject;
    }
    public Object invoke(Object proxy, Method method,
                        Object[] args) throws Throwable {
        return method.invoke(getSubject(), args);
    }
}
```

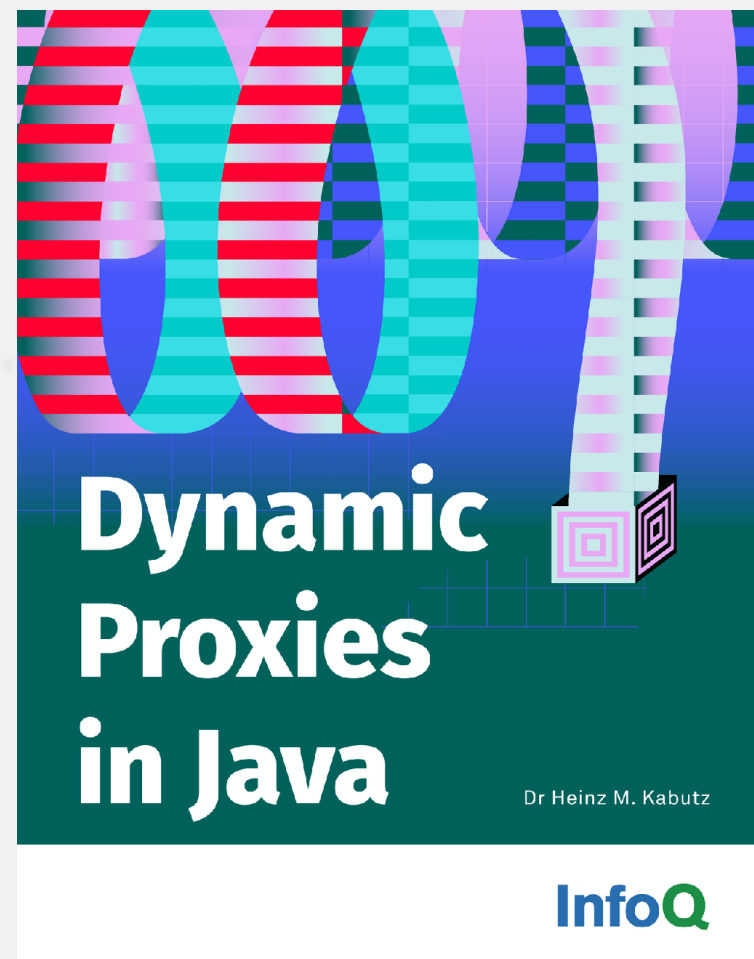
Proxies Facade virtualProxy()

- Facade has a `virtualProxy()` method

```
public static <S> S virtualProxy(  
    Class<? super S> subjectInterface,  
    Supplier<? extends S> subjectSupplier) {  
    Objects.requireNonNull(subjectSupplier,  
        "subjectSupplier==null");  
    return castProxy(subjectInterface,  
        new VirtualProxyHandler<>(subjectSupplier));  
}
```

- More details in book

– www.infoq.com/minibooks/java-dynamic-proxies/



Creating Virtual Proxy

- We can create virtual proxies of anything
 - Here we replace the handcrafted proxy with dynamic
 - Less code, less chance of bugs

```
CustomMap<String, Integer> map =  
    Proxies.virtualProxy(CustomMap.class,  
        CustomHashMap::new);  
System.out.println("Virtual Map created");  
map.put("one", 1); // creating map as side effect  
map.put("life", 42);  
System.out.println("map.get(\"life\") = " +  
    map.get("life"));  
System.out.println("map.size() = " + map.size());  
System.out.println("clearing map");  
map.clear();  
System.out.println("map.size() = " + map.size());
```

```
Virtual Map created  
CustomHashMap constructed  
map.get("life") = 42  
map.size() = 2  
clearing map  
map.size() = 0
```

Dynamic Proxies in Java

Dynamic Proxy Restrictions



Javaspecialists.eu
java training

Interfaces Only

- **Dynamic proxies cannot extend classes**
 - All proxies are subclasses of `java.lang.reflect.Proxy`
 - No multiple inheritance in Java
 - Might need to use tools like **CGLib** or **ByteBuddy**

UndeclaredThrowableException

- **InvocationHandler.invoke()** throws **Throwable**
 - However, we should only throw declared exceptions
 - **Error and RuntimeException** always allowed

```
Runnable job = Proxies.castProxy(
    Runnable.class,
    (proxy, method, params) -> {
        // will be wrapped with UndeclaredThrowableException
        throw new IOException("bad exception");
    });
job.run();
```

```
Exception in "main" java.lang.reflect.UndeclaredThrowableException at
com.sun.proxy.$Proxy0.run(Unknown Source)
at UndeclaredExceptionThrown.main()
Caused by: java.io.IOException: bad exception
at UndeclaredExceptionThrown.lambda$main$0() ... 2 more
```

Return Types Have to be Correct

```
public interface FooBar {
    void foo();
    boolean bar();
    int baz();
}

public class FooBarInvocationHandler
    implements InvocationHandler {
    public Object invoke(Object proxy, Method method,
                        Object[] args) throws Throwable {
        switch (method.getName()) {
            case "foo": return true; // ignored
            case "bar": return 42; // ClassCastException
            case "baz": return null; // NullPointerException
            default: throw new UnsupportedOperationException();
        }
    }
}
```

Naming Mysteries

```
public class ProxyNaming {
    public interface PublicNotExported {
        void open();
    }
    interface Hidden {
        void mystery();
    }
    public static void main(String... args) {
        show(BaseComponent.class); // exported from module
        show(PublicNotExported.class);
        show(Hidden.class);
    }
    private static void show(Class<?>... intf) {
        System.out.println(Proxy.newProxyInstance(
            intf[0].getClassLoader(), intf,
            (p, m, a) -> null).getClass());
    }
}
```

```
class com.sun.proxy.$Proxy0
class com.sun.proxy.jdk.proxy1.$Proxy1
class eu.javaspecialists.books.dynamicproxies.ch03.gotchas.$Proxy2
```

Deeper Call Stacks

- **Call stacks have references to \$Proxy0**
 - Can increase each call by an additional 5 layers
 - IntelliJ IDEA folds this information away

```
micProxy$Factorial.invoke(RecursiveDynamicProxy.java:46) <5 interna  
micProxy$Factorial.invoke(RecursiveDynamicProxy.java:49) <5 interna  
micProxy$Factorial.invoke(RecursiveDynamicProxy.java:49) <5 interna  
micProxy$Factorial.invoke(RecursiveDynamicProxy.java:49) <5 interna
```

- **Actual stack trace contains all the gory details**

```
at RecursiveDynamicProxy$Factorial.invoke()  
at com.sun.proxy.$Proxy0.apply(Unknown Source)  
at java.base/NativeMethodAccessorImpl.invoke0()  
at java.base/NativeMethodAccessorImpl.invoke()  
at java.base/DelegatingMethodAccessorImpl.invoke()  
at java.base/Method.invoke()  
at RecursiveDynamicProxy$Factorial.invoke()
```

Shared Proxy Classes

- **Java tries to minimize dynamic proxy classes**
 - **When we call `Proxy.newProxyInstance()` it checks**
 - **Have we had the same interfaces (in that order)?**
 - **And for the same class loader?**
 - **If it has seen it before, it returns a cached class**
 - **This is stored inside weak references to prevent memory leaks**

Dynamic Proxies in Java

Performance



Javaspecialists.eu
java training

Performance

- **Dynamic proxies used in infrastructure code**
 - **Some methods called billions of times**
- **Calling methods on dynamic proxies may be slower**
 - **Primitive return types and parameters might be boxed**
 - **Parameters are wrapped with Object[]**
 - **Object[] can be eliminated if it does not escape from invoke()**
 - **Method has amnesia and checks our permission every call**

Model for Benchmark using JMH

```
public interface Worker {  
    long increment();  
    void consumeCPU();  
}  
  
public class RealWorker implements Worker {  
    private long counter = 0;  
  
    public long increment() { return counter++; }  
    public void consumeCPU() { Blackhole.consumeCPU(2); }  
}  
  
public class ProxyWorker implements Worker {  
    private final RealWorker worker = new RealWorker();  
  
    public long increment() { return worker.increment(); }  
    public void consumeCPU() { worker.consumeCPU(); }  
}
```


increment() and consumeCPU()

- **We use five ways of calling these methods**
 - **directCall**
 - **staticProxy**
 - **dynamicProxyThenDirectCall**
 - **Avoids cost of reflective method calls, but might need to box return value**
 - **dynamicProxyThenReflectiveCall**
 - **Delegates all calls to the RealWorker using reflection**
 - **Method call might be turbo-boosted by turning off accessibility checks**

Benchmark increment() Results

- **Analysis of results**

- **dynamicProxyDirectCall 2.1 ns slower than staticProxy**
- **dynamicProxyReflectiveCall is another 4.1 ns slower**
 - **Also allocates 24 bytes**
- **Without our method turbo boost, it is another 2.3 ns slower**

| Benchmark increment() | Best ns/op | Bytes/op EA on/off |
|--|-------------------|---------------------------|
| directCall | 2.9 | 0 / 0 |
| staticProxy | 3.5 | 0 / 0 |
| dynamicProxyDirectCall | 5.6 | 0 / 24 |
| dynamicProxyReflectiveCall (turbo) | 9.7 | 24 / 24 |
| dynamicProxyReflectiveCall (no turbo) | 12 | 24 / 24 |

Benchmark consumeCPU() Results

- **Analysis of results**

- **dynamicProxyDirectCall 1.1 ns slower than staticProxy**
- **dynamicProxyReflectiveCall is another 1 ns slower**
- **Without our turbo boost, it is a further 3.4 ns slower**

| Benchmark consumeCPU() | Best ns/op |
|--|------------|
| directCall | 4.8 |
| staticProxy | 5.5 |
| dynamicProxyDirectCall | 6.6 |
| dynamicProxyReflectiveCall (turbo) | 7.6 |
| dynamicProxyReflectiveCall (no turbo) | 11 |

Summary of Benchmark Results

- **Method call overhead for our experiments**
 - 6.2 nanoseconds for increment()
 - 2.1 nanoseconds for consumeCPU()
- **Overheads negligible in typical business application**
 - Unless called in performance sensitive code

3: Related Patterns



3: Related Patterns

- **Proxy has a similar structure to**
 - **Decorator / Filter**
 - **Composite**

Filtering for Immutability



ImmutableCollection

```
public interface ImmutableCollection<E> {  
    void forEach(Consumer<? super E> action);  
    Spliterator<E> spliterator();  
    int size();  
    boolean isEmpty();  
    boolean contains(Object o);  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
    <T> T[] toArray(IntFunction<T[]> generator);  
    boolean containsAll(Collection<?> c);  
    Stream<E> stream();  
    Stream<E> parallelStream();  
    // to try out default methods  
    default void printAll() {  
        forEach(System.out::println);  
    }  
}
```


HandcodedFilter

- **We only offer methods that do not mutate**

```
public class HandcodedFilter<E>
    implements ImmutableCollection<E> {
    private final Collection<E> c;

    public HandcodedFilter(Collection<E> c) { this.c = c; }

    public int size() { return c.size(); }
    public boolean isEmpty() { return c.isEmpty(); }
    public boolean contains(Object o) { return c.contains(o); }
    public Object[] toArray() { return c.toArray(); }
    // etc.
    // No mutable methods are offered to client
}
```

FilterHandler

```
public final class FilterHandler
    implements InvocationHandler {
    private final ChainedInvocationHandler chain;

    public FilterHandler(Class<?> filter, Object component) {
        VTable vt = VTables.newVTable(component.getClass(), filter);
        VTable defaultVT = VTables.newDefaultMethodVTable(filter);

        chain = new VTableHandler(component, vt,
            new VTableDefaultMethodsHandler(defaultVT, null));
        chain.checkAllMethodsAreHandled(filter);
    }
    public Object invoke(Object proxy, Method method, Object[] args)
        throws Throwable {
        return chain.invoke(proxy, method, args);
    }
}
```

Dynamic Filter

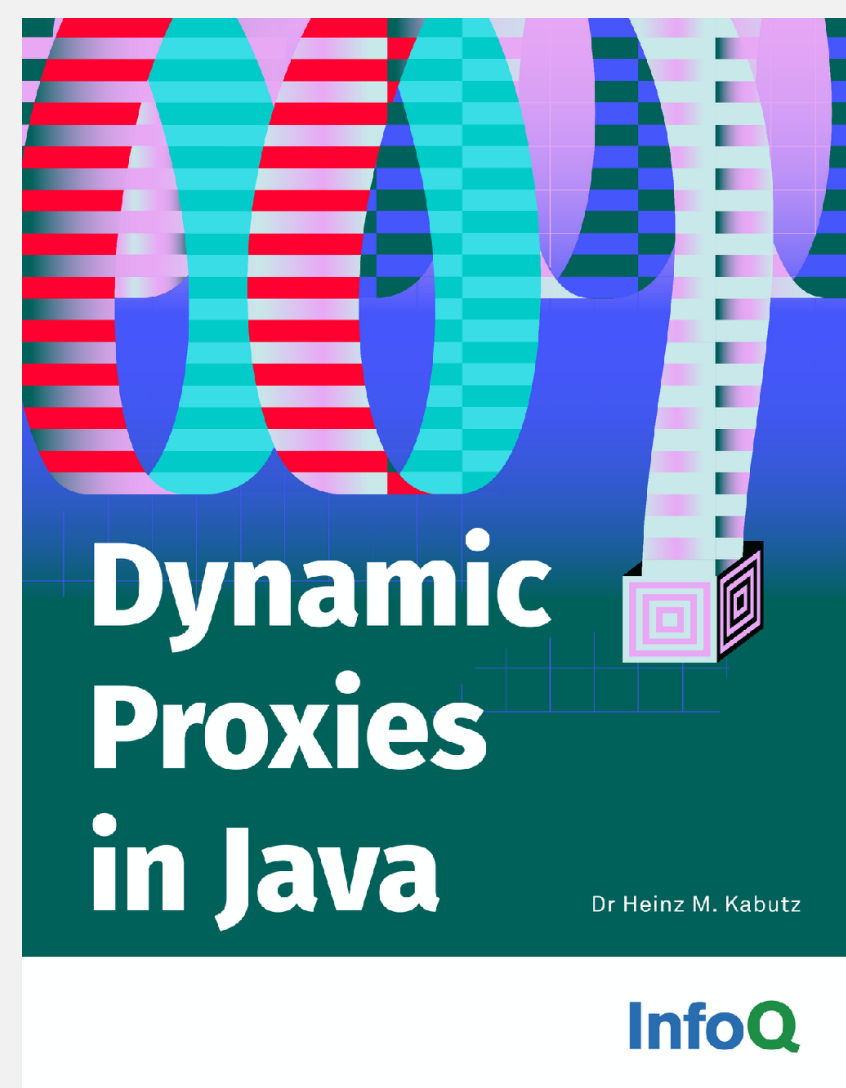
- We can filter away the mutating methods

```
ImmutableCollection<String> names =  
    Proxies.filter(ImmutableCollection.class,  
        Arrays.asList("Peter", "Paul", "Mary"));  
// names.remove("Peter"); // does not compile  
System.out.println(names);  
System.out.println("Is Mary in? " +  
    names.contains("Mary"));  
System.out.println("Class: " +  
    names.getClass().getSimpleName());  
names.printAll();
```

```
[Peter, Paul, Mary]  
Is Mary in? true  
Class: $Proxy0  
Peter  
Paul  
Mary
```

Questions?

- Don't forget gift: tinyurl.com/nairobi-jug
- Free Java Specialists' Newsletter
 - www.javaspecialists.eu/archive/subscribe.jsp
- Please say "hello" : heinz@javaspecialists.eu



- Free download from
 - <https://www.infoq.com/minibooks/java-dynamic-proxies/>